Advanced Analysis of Branch and Bound Algorithms

Marcel Turkensteen

Publisher: Labyrint Publications
P.O. Box 334
2984 AX Ridderkerk
The Netherlands
Tel: 0180-463962

Printed by:  Offsetdrukkerij Ridderprint B.V., Ridderkerk

ISBN-10: 90-5335-110-8
ISBN-13: 978-90-5335-110-9

RIJKSUNIVERSITEIT GRONINGEN

ADVANCED ANALYSIS OF BRANCH AND BOUND ALGORITHMS

Proefschrift

ter verkrijging van het doctoraat in de
Economische Wetenschappen
aan de Rijksuniversiteit Groningen
op gezag van de
Rector Magnificus, dr. F. Zwarts,
in het openbaar te verdedigen op
donderdag 15 februari 2007
om 13:15 uur

door

Marcel Turkensteen

geboren op 18 januari 1979
te Tolbert

Promotor:	Prof. dr. G. Sierksma
Copromotores:	Dr. B. Goldengorin Dr. D. Ghosh
Beoordelingscommissie:	Prof. dr. G. Gutin Prof. dr. W.K. Klein Haneveld Prof. dr. M. Wedel

ISBN-10: 90-5335-110-8
ISBN-13: 978-90-5335-110-9

Acknowledgements

Many people have made contributions to this dissertation. I would like to express my gratefulness towards them here.

Firstly, I would like to thank my promoter, Gerard Sierksma. He gave me the freedom to do my work my own way, and I enjoyed that freedom. We also had many interesting and stimulating discussions during our weekly meetings. Moreover, I have learned a lot from working with Gerard. He taught me about doing research and about reporting the results. I also got the chance to grow as a person under his supervision.

Secondly, I would like to thank my co-promoters, Diptesh Ghosh and Boris Goldengorin. To start with Boris, I appreciate his enthusiasm and the discussions about the research. Diptesh supervised me mainly from a distance, but working with him was fun and very inspiring, especially during his visits to Groningen.

Next, I am grateful to the members of the reading committee, Gregory Gutin, Wim Klein Haneveld, and Michel Wedel, and thank them for their interest in my dissertation and for their useful comments.

I would also like to acknowledge the people I co-operated with: Jaap Wieringa, Douwe Postmus, Remco Germs, Lolke Schakel, Bertus Talsma, Gerold Jäger, and Gerco Brands. Jaap co-authored Chapter 5 with me and provided some very valuable insights from the field of marketing research. Douwe did splendid exploratory work for Chapter 5. Gerold and Remco performed related research on upper and lower tolerance-based algorithms. Lolke and Bertus were my office mates during most of my PhD. I not only shared an office with them, but also a lot of discussion and fun.

I enjoyed working at the Marketing Department with its ambitious people and team spirit. I am not going to list all members individually, but I would specially like to thank Jeannette and Hanneke of the secretary, in particular for their help and company. I am also grateful to the members of the Department of Econometrics for giving me a place among them during the last year of my PhD. I thank all the people in the helpdesk, support services, and to everyone who helped to improve my skills.

Believe it or not, there was life outside the office. I would like to express my gratefulness to my friends for giving me a good time here. I am very grateful for

my family, for always being there when I needed it.

Last but not least, I would like to thank Jos and Ernst for supporting me during my defense ceremony, and for wearing penguin suits with me.

Groningen, February 2007
Marcel Turkensteen

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Advanced Analysis of Branch and Bound: the title suggests that this thesis deals with a subject that is disconnected from daily life. However, the problems studied in this thesis are frequently encountered in practice. To give the reader a feel for the type of problems discussed in this dissertation, this introduction starts gently with two practical and easily understandable applications, namely Google and route planners. In this thesis, we do not consider these specific applications. Instead, we consider some similar practical problems that are computationally very difficult to solve. Solution times are typically so long, that it is a major challenge to improve methods for obtaining optimal solutions to such problems, such as *Branch and Bound methods*.

When we browse the internet, it is very likely that we use a search engine such as Google. This application searches through billions of web sites and produces a set of results in less than a second. For example, a search for the term 'Groningen' yields about 43 million results in just 0.5 seconds.[1] Moreover, it makes an ordering of the websites in such a way that the most relevant one is presented first. How can Google operate so quickly?

Other popular internet applications are route planners. Suppose that a person uses public transportation to travel from location A to location B, for example from the train station to a university building. The travel schedule can be determined on the internet, for example, on `http://www.9292ov.nl` in the

[1] Search on May 9, 2006, 11:37 AM

Netherlands. Given a starting location A, a destination B and an arrival or a departure time, the computer determines the fastest schedule, and it does so within a second. However, the system contains a huge database of addresses and public transport connections. So how does the system retrieve the fastest trajectory so quickly?

In this thesis we study a category of problems that occur in a wide range of applications, the so-called *optimization problems*. In these problems, we search through a set of candidate solutions to find a solution that satisfies an *a priori* objective. One of the most well-known problems is the *shortest path problem*, where one wants to find a shortest path between two given points. The *candidate solutions* are the paths between the two locations and the a priori objective is 'determine a shortest path'. This problem is an example of a so-called Combinatorial Optimization Problem, or COP, because one wants to find a *combination* of such segments with minimal length. A formal definition of a general COP is given in Section 1.2.

For the so-called *easy* COPs, quick solution methods exist. For example, route planners are able to determine a shortest path between two locations almost instantaneously, even when the number of possible routes is huge. On the other hand, there is still a large class of *hard* problems for which efficient and quick methods that solve all instances to optimality are not available yet. Exhaustive search is needed for such problems, meaning that, in principle, all possible options need to be checked. Not only scientists, also decision makers in industry and in governments face such difficult problems frequently. In modern decision making, large quantities of data are available to support the decision. Effective processing of the information may help, for example, to serve customers with tailored offers, or to implement more effective schedules in factories. Improving solution methods for these problems is of great practical importance, even when seemingly small improvements are made.

The contribution of this dissertation lies in the improvement of solution methods of difficult COPs to optimality. First of all, a definition of COPs is provided. After that, are view of *complexity theory* is given, i.e., the field of research that studies the complexity of COPs. Next, a short overview is given of methods for solving difficult problems to optimality.

This dissertation consists of four separate papers about a wide range of sub-

jects in *Branch and Bound*, a class of methodes explained in Section 1.5. Chapter 2, 3, and 4 discuss potential improvements to the BnB techniques themselves. Chapter 5 considers an application of these techniques in the field of marketing.

1.2 Combinatorial Optimization Problems

In this section, a formal definition of a *Combinatorial Optimization Problem (COP)* is given. According to Wikipedia (2006), "combinatorial optimization is a branch of optimization in applied mathematics and computer science, related to operations research, algorithm theory and computational complexity theory that sits at the intersection of several fields, including artificial intelligence, mathematics and software engineering." In Goldengorin (2002), a COP is defined by the following quadruple $(\mathcal{E}, \mathcal{D}, C, f_C)$. The set of elements of a COP is called the *ground set* and it is denoted by $\mathcal{E}$. The set $\mathcal{D}$ contains the feasible solutions that can be constructed from the elements in $\mathcal{E}$; $\mathcal{D}$ should be finite (Schrijver, 2003). The feasible solutions are evaluated with the *objective function* f_C. The *additive* cost function, used throughout most of this dissertation, adds the costs of all elements in a chosen combination. By 'costs' we do not only mean monetary costs, but also distances, times, weights, or any other measure reflecting our objective function. Finally, C denotes the specific *instance* of the problem. For instance, a shortest path instance is characterized by the distances between each pair of locations.

Many COPs are graph problems, occurring in, for example, transportation or communication networks. Since terms from graph theory are used throughout this dissertation, a basic introduction into graph theory is given here. A *graph* $G(V, E)$ consists of a set of vertices V and a set of edges E between the vertices. If the edges are directed, the graph $G(V, A)$ is called a *digraph* with the set of vertices V and the set of *arcs* A. A *walk* or a *path* is alternating sequence of arcs or edges and vertices between a pair of vertices $v, w \in V$. A *tour* or a *cycle* is a closed walk, meaning that the starting point and the end point are the same. A *tree* on a subset $V' \subseteq V$ of the vertices is a set of edges such that V' is connected but there are no cycles in V'. Figure 1.1 shows examples of a tree and a tour. An extensive introduction in graph theory is given in, among others, Jungnickel (2005) and Bang-Jensen and Gutin (2001).

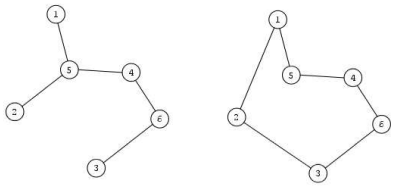

Figure 1.1. Examples of a tree (left) and a tour (right)

Take for example the shortest path problem solved by route planners. The ground set is formed by the arcs between each pair of locations, and the set of solutions contains all possible paths between the origin and the destination. The instance is defined by the costs of all the arcs, and the objective is minimization of the path length.

1.3 Complexity theory

Why can the easy problems, such as those in Google and route planners, be solved quickly, whereas other problems are very difficult to solve? The field of research that studies this question is *complexity theory*.

The most well-known example of a difficult COP is the *Traveling Salesman Problem (TSP)* (Punnen, 2002), which is defined as follows. Given n locations and given the matrix C containing the distances between each pair of locations, find the shortest tour through all locations such that each location is visited exactly once. An extensive list of applications of the TSP is given in Punnen (2002). The TSP is easy to formulate intuitively, and at first sight, it may appear straightforward to solve. However, the appearance is deceptive, since in the worst case, all solutions of the problem must be enumerated. The TSP is the most widely known hard problem, but there are many other ones.

The number of possible combinations in a COP usually increases very rapidly with the number of elements from which the combinations are chosen, the *input size*; see Garey and Johnson (1979). For example, there may be as many as $(n-1)!$ possible TSP solutions through n cities. So when $n = 20$, there are

approximately $19! \approx 1.216 \times 10^{17}$ possible tours. In a complete network with n intermediate locations, there are approximately $(n-2)!$ possible paths from which a shortest one should be chosen. If $n = 20$, there are potentially as many as 6.40×10^{15} paths. In the examples at the beginning of this introduction, the number of websites and the number of possible routes between a pair of locations are even much larger. For most COPs, the number of solutions increases exponentially with the size of the input n, meaning that the number of solutions for input size $n+1$ is typically a multiple of the number of solutions for input size n (Garey and Johnson, 1979).

Is it necessary to investigate all, or a large part of these possible solutions, or does it suffice to consider only a small fraction? In other words, when is a problem easy or difficult to solve? The hardness varies tremendously between COPs: some instances of easy problems of size 100,000 are solved within seconds, whereas it may take a long time to solve some hard problem instances of size 100, if they can be solved at all. For example, the notoriously hard Quadratic Assignment Problem (QAP) is the problem of assigning n facilities to n locations, in such a way that the sum of the weights of the flow between each pair of facilities times the distance between them is minimized. The largest non-trivial QAP instances solved to optimality are of size 32 (Loiola *et al.*, 2007).

The difficulty of a COP is usually measured with the *worst-case time complexity* of the fastest algorithm solving it to optimality (Aho *et al.*, 1974). An *algorithm* is defined in Knuth (1997) as a procedure with four characteristics: finite running times, precisely defined steps, input and output.

The worst-case complexity of a problem is written as a function $f(n)$ of the input size n of the problem instance; see for example (Cook *et al.*, 1998). The input size contains the number of elements in the ground set. The function $f(n)$ contains the number of basic operations, such as adding, subtracting or comparing numbers, that have to be carried out to find an optimal solution of a problem with problem size n. The expression $O(f(n))$, which should be read 'of order $f(n)$', is used to denote the complexity of a COP. Let N denote the number of operations, then $f(n)$ is the smallest function such that $N \leq Cf(n)$ with C being a constant. For example, if the fastest algorithm for a COP needs at most $3n^2 + 15n$ operations, then the COP is $O(n^2)$, since $3n^2 + 15n < 4n^2$ for $n > 15$. *Polynomial algorithms* are of polynomial order, for example $O(n^2)$

or $O(n^3 \log(n))$, whereas *exponential algorithms* are of higher order, for example $O(2^n)$ or $O(n!)$. Table 1.1 shows that the solution times of exponential algorithms increase much more rapidly than the solution times of polynomial algorithms. The reported times are hypothetical; it is assumed that a computer carries out one million elementary operations per second. Since the number of operations of polynomial algorithms increases relatively slowly with the input size n, these algorithms are called *efficient* (Schrijver, 2003). For an *easy* COP, efficient algorithms are available, and the worst-case complexity is polynomial. On the other hand, the worst-case time complexity of a problem for which only exponential algorithms are available, is exponential, which can make the problem difficult to solve. An extensive account on complexity theory is given in Garey and Johnson (1979).

Table 1.1. Speed of polynomial and exponential time algorithms (Garey and Johnson, 1979)

Time complexity function	Size n 10	30	50
n	0.00001 sec.	0.00003 sec.	0.00005 sec.
n^2	0.0001 sec.	0.0009 sec.	0.0025 sec.
n^5	0.1 sec.	24.3 sec.	5.2 min.
2^n	0.001 sec.	17.9 min	35.7 years
3^n	0.59 sec.	6.5 years	2×10^8 centuries

The previous part considered worst-case complexities to measure the difficulty of a COP. However, the fact that an algorithm is exponential does not automatically imply that it is always slow: an $O(2^n)$ algorithm may be able to solve most problems in polynomial time, requiring exponential time only for a small subset of instances. A well-known example of such an algorithm is the *Simplex-method* from Dantzig *et al.* (1955). This popular algorithm needs a polynomial number of operations for many Linear Programming instances (Todd, 2002; Wolfe and Cutler, 1963), but there are some instances for which exponential time is needed (Klee and Minty, 1972). The performance of the algorithms in this dissertation is therefore mainly measured with *average case analysis* instead of worst-case analysis. This means that solution methods are tested on a wide range of typical instances and average solution times are measured.

Time complexity of COPs is a widely studied field of research, the most no-

table work being Cook (1971); Karp (1972, 1975); Garey and Johnson (1979). The main result from Cook (1971) deals with decision problems, i.e., problems with yes or no as answers. For example, a TSP can be expressed in the decision form as follows: is there a complete tour through n locations with cost at most c, given a cost matrix C? Cook (1971) classifies all such problems which have a yes answer verifiable in polynomial time as the class $\mathcal{NP}$. This class $\mathcal{NP}$ consists of two broad classes. The first class is called polynomial ($\mathcal{P}$). Problems in these class are those, which, in addition to being in the class $\mathcal{NP}$, are also easy to solve, i.e., there exist polynomial algorithms for problems in this class. Examples of such problems are the shortest path problems, minimum spanning tree problem and the assignment problem. The second class of problems in $\mathcal{NP}$ are said to belong to the class $\mathcal{NP}$-Complete. A problem is said to belong to the class $\mathcal{NP}$-Complete, if a polynomial time algorithm to solve the problem would lead to a polynomial time algorithm to solve every problem in the class $\mathcal{NP}$. Examples of problems in this class are the TSP and the QAP. Unfortunately, we do not have polynomial time algorithms to any problem in the $\mathcal{NP}$-Complete class. The question as to whether $\mathcal{P}$= $\mathcal{NP}$ remains one of the most well-known problems in Computer Science to date. In fact, it belongs to the seven so-called *Millennium Prize Problems*[2], which have a reward of $1,000,000 (Clay Mathematics Institute, 2006).

The $\mathcal{NP}$-Complete class of problems are defined for decision problems. In our study we deal with optimization versions of problems. For example, the optimization version of a TSP is the following. Find a shortest complete tour through n locations, given a cost matrix C. The optimization version of $\mathcal{NP}$-Complete problems are said to be $\mathcal{NP}$-hard problems.

Frequently encountered $\mathcal{NP}$-hard problems are the Traveling Salesman Problems, Facility Location Problems (Goldengorin *et al.*, 2004), and Integer Programming Problems (Cook *et al.*, 1998). The clustering problems from Chapter 5 are also $\mathcal{NP}$-hard; see Garey and Johnson (1979). An up-to-date list of $\mathcal{NP}$-hard problems is maintained in Kann and Crescenzi (2006).

[2]One of the seven problems, the Poincaré Conjecture, has been solved by Grigori Perelman.

1.4 Solving $\mathcal{NP}$-hard problems

We have seen that it can be very time-consuming to solve some large $\mathcal{NP}$-hard problem instances to optimality. Should we focus on improving quick methods that may return suboptimal solutions, or should we try to improve exact methods?

Heuristics are algorithms which return non-optimal solutions for some instances of a problem. Much recent scientific work is done on heuristics; see, for example, Affenzeller and Mayrhofer (2002). An important cause of the recent focus on heuristics is the emerging of many *online applications*, in which a decision must be taken almost instantaneously (González *et al.*, 2001). Since heuristics typically have short solution times, they are employed in online optimization.

A promising class of heuristics is called *meta-heuristics* (Glover and Kochenberger, 2003): solution methods that orchestrate an interaction between local improvement procedures and higher strategies, such as random construction of new solutions, to create a process capable of escaping from local optima. The problem with improvement heuristics is that they may become trapped in a local optimum; see Figure 1.2. A meta-heuristic uses strategies to escape from such a local optimum. The simulated annealing algorithm presented in Chapter 5 is a meta-heuristic. Other prominent classes of meta-heuristics are genetic algorithms, tabu search and variable neighborhood search (Glover and Kochenberger, 2003). Meta-heuristics are successfully applied on a wide range of problems, such as the TSP (Buriol *et al.*, 2004) and clustering problem (DeSarbo and Grisaffe, 1998). Solutions with at most 1% higher cost than the optimal solution are often obtained.

Research on exact methods certainly remains worthwhile. There are many problems in which the difference of 1% in solution value leads to millions of euros or dollars of extra costs compared to an optimal solution. This holds in particular for *strategic decision making*, where resources are allocated for long periods of time, such as the segmentation decisions from Chapter 5. Clearly, it is more important to support the decision accurately than to obtain a solution quickly. Improvements in exact methods enable us to solve problems larger and more difficult problem instances to optimality.

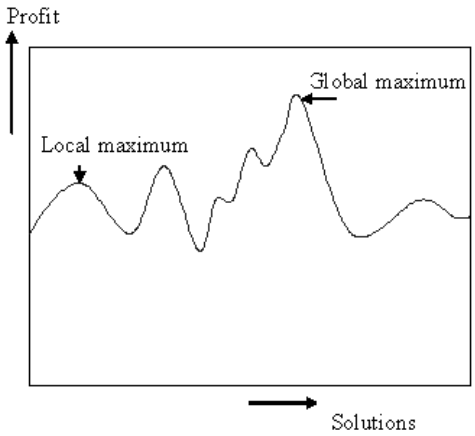

Figure 1.2. Local and global optima

Research on heuristics and exact algorithms can reinforce each other. In some applications, (meta-)heuristics and exact methods are combined into effective new algorithms; see for example Cotta and Troya (2003); Nwana *et al.* (2005) and Chapter 2.

The most frequently used type of exact methods for solving $\mathcal{NP}$-hard COPs is Branch and Bound (BnB); the next section is devoted this type of methods. Many variants of BnB exist, such as Branch and Cut (Naddef, 2002), Branch and Price, Branch and Peg (Goldengorin *et al.*, 2004), Branch and Win (Pastor and Corominas, 2004), and Cut and Solve (Climer and Zhang, 2006). Other exacts methods are Dynamic Programming, cutting plane methods, complete enumeration, and Data Correcting (Goldengorin, 2002).

1.5 The Branch and Bound Methodology

In this Section, the Branch and Bound (BnB) methodology is discussed. BnB methods have been applied successfully on various $\mathcal{NP}$-hard problems, such as Traveling Salesman Problems (Miller and Pekny, 1991; Carpaneto *et al.*, 1995), Mixed Integer Programming problems (Achterberg *et al.*, 2004; Sierksma, 1996), and Scheduling Problems (Baptiste *et al.*, 2004). State-of-the-art BnB methods are able to solve large instances to optimality. For example, the exact algorithm from Applegate *et al.* (2004) solved a non-trivial symmetric TSP instance con-

sisting of 33,810 locations in 2005 (Wikipedia, 2006).

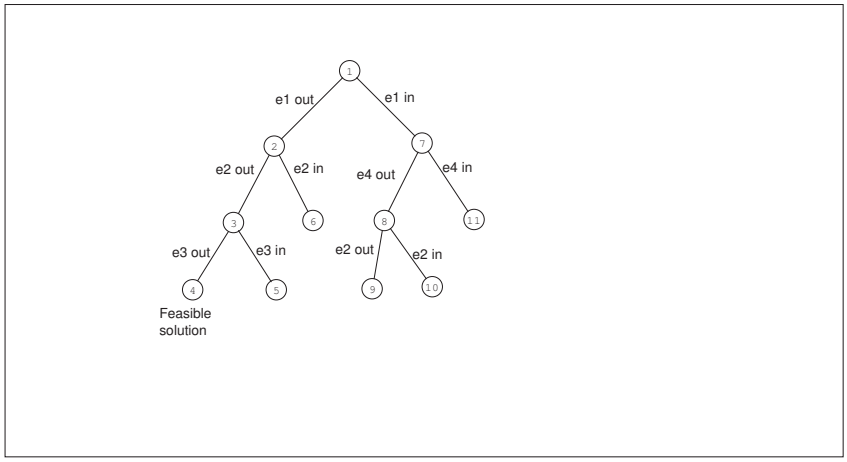

Figure 1.3. Example of a BnB search tree

The following phenomenon frequently occurs in Combinatorial Optimization: if certain constraints of the original hard problem are removed, the problem becomes easily solvable; see for example Schrijver (2003). Such a less constrained problem is called a *relaxation*. There are numerous examples of $\mathcal{NP}$-hard problems with easily solvable relaxations, the most famous example being Integer Programming and its Linear Programming relaxation.

BnB methods use easily solvable relaxations as follows. Initially, an easy relaxation is solved. If the solution obtained is feasible for the original hard problem, then this solution is optimal for the original problem as well and the BnB method for the partivular problem terminates; otherwise, the problem is divided into smaller and more restricted problems: the *subproblems*. The process continues until all subproblems are either solved or *discarded*, meaning that the subproblem is not relevant for determining an optimal solution. Another term for discarding is *fathoming*. BnB methods list all solutions in a search tree; see Figure 1.3. We refer to Ibaraki (1987) for a detailed description of the BnB process.

A BnB method for a minimization problem is characterized by the following four building blocks; see Miller and Pekny (1991):

Upper bound. Usually, the upper bound is the value of the best solution obtained

so far.

Lower bound. The lower bound of a subproblem should be smaller than or equal to the smallest value of any feasible solution of that subproblem. If a lower bound exceeds the value of the upper bound, we discard the corresponding subproblems.

Branching rule. At each node of a BnB search tree, a branching rule prescribes how the current problem should be partitioned into new subproblems.

Search strategy. The search strategy prescribes how the BnB algorithm should proceed through the search tree. The two most common methods are *Depth First Search*, which solves the most recently generated subproblem first, and *Best First Search*, which solves the most promising subproblem first.

A BnB algorithm for a maximization problem is similar, but the roles of upper bounds and lower bounds are exchanged.

1.6 Improvements to Branch and Bound methods

It is often thought that improvements on solution techniques are mainly due to increasing computing power, but this is not quite true. For example, Bixby (1994) finds that in the period 1980-1990, algorithmic improvements are the main cause of solution time reductions in Linear Programming. One cannot rely solely on increasing computing power; improvements in BnB algorithms are needed to solve large instances of $\mathcal{NP}$-hard problems to optimality.

This dissertation aims to improve the building blocks of BnB algorithms as follows:

- A subproblem is fathomed if the value of its lower bound exceeds the value of the best solution obtained so far, the *upper bound*. The more subproblems are fathomed, the smaller the search trees. Can we improve the upper bound, and if yes, does the time invested in such an improvement decrease total solution times? Chapter 2 describes the concept of *iterative patching*, a procedure to construct effective upper bounds for the Asymmetric Traveling Salesman Problem.

- Search tree reductions are achieved by decreasing the upper bounds, but also by increasing the lower bounds of subproblems. To this end, we increase the lower bound of each subproblem with so-called *upper tolerances* in Chapter 3 and 4. Again, we face the trade-off between the improvement in lower bounds and the time it takes to determine the bounds.

- At each node of a BnB search tree, the *branching rule* prescribes how the current problem should be partitioned into subproblems. A branching rule is effective if it preserves elements or structures which are also in an optimal solution of the hard problem and deletes the other ones. We introduce new *tolerance-based branching rules* in Chapter 3. We try to improve these branching rules in Chapter 4.

1.7 Clustering and market segmentation

In marketing, a company can distinguish itself from its competitors by offering each relevant group of customers a tailored marketing mix (Wedel and Kamakura, 1998). A group of consumers is only targeted well if the consumers in the group achieve similar scores on the relevant attributes, such as price sensitivity or income. They should respond more or less the same on the marketing efforts of the company, such as a price increase.

The grouping of subjects in similar groups is the so-called *Clustering Problem*. Clustering is, according to the definition from Mirkin and Muchnik (1998), "a mathematical technique for revealing classifications in the data collected on real world phenomena". Clustering Problems are encountered in a wide range of disciplines, such as computer science (Abrantes and Marques, 1996) and pattern recognition (Pawitan and Huang, 2003), but we concentrate on an application in market research. The clustering of consumers in marketing is called *market segmentation*.

In Steenkamp and Ter Hofstede (2002), it is noted that in many international segmentations studies, logistics costs are so high that organization resort to the segmentation strategy in which countries are taken as segments. The resulting marketing offers of the company fit consumer preferences poorly. In Chapter 5, the trade-off between logistics costs and the fit of consumer preferences is

explicitly made. To obtain such segmentations, we use meta-heuristics and BnB algorithms.

Chapter 2

Iterative Patching and the Asymmetric Traveling Salesman Problem

2.1 Introduction

The Asymmetric Traveling Salesman (ATSP) is usually solved exactly by means of Branch-and-Bound (BnB) algorithms and Branch-and-Cut (BnC) algorithms, see Fischetti *et al.* (2002). In BnB type algorithms, an Assignment Problem (AP) is solved at every node of this tree, and the value of the optimal AP solution serves as a lower bound of the ATSP solution. A part of the search tree can be discarded when its lower bound exceeds an upper bound. This upper bound is usually the value of a shortest complete tour found so far. A class of heuristics applied to construct such a tour is *patching*. The question is: at which nodes of the search tree should such a tour be constructed? Patching at a node may reduce the search tree and the solution time, but if the reduction is too small, the overall solution time is increased due to the time invested in patching.

In the literature, the most effective BnB methods do not patch at each node; see for example, Miller and Pekny (1991), Carpaneto *et al.* (1995). These methods use a best first search strategy, i.e., the subproblem with the smallest lower bound is solved first. According to these studies, patching at every node is too

[0] Joint work with D. Ghosh, B. Goldengorin and G. Sierksma. Published in Discrete Optimization (2006), issue 3 (1), p. 63–77.

time-consuming.

In this paper, we consider a BnB algorithm that applies depth first search, which means that the most recently generated subproblem is solved first. This strategy requires algorithms to use much less computer memory than do best first strategies. Hence, it is useful for solving large problems. We apply *iterative patching* , in which a fixed patching procedure is applied at every node of the BnB depth first search tree. Four iterative patching procedures are considered in our computational experiments. These procedures are described in Glover *et al.* (2001).

Given a set of locations and the distance between any pair of locations, the ATSP is the problem of finding a shortest Hamiltonian tour; i.e., a shortest round trip visiting each location exactly once. Figure 2.1 is an example of an underlying graph that defines an instance of an ATSP. The nodes of the graph represent locations, and the arcs the connections between the locations. A number next to an arrowhead denotes the cost of traveling along that arc.

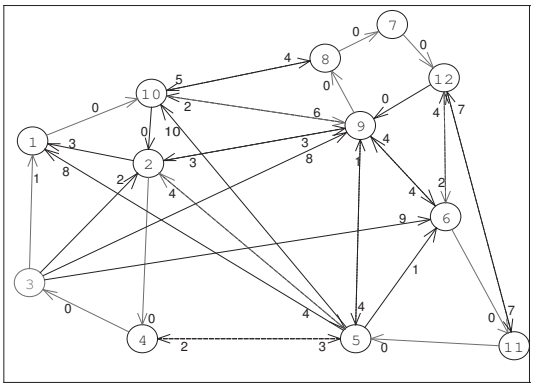

Figure 2.1. ATSP instance

General instances of the ATSP are often solved to optimality by means of enumeration algorithms, in which a fraction of all feasible solutions is checked. BnB methods explore the solution space by using a search tree. We discuss BnB algorithms that solve an Assignment Problem (AP) at each node of the corresponding search tree. After solving the AP a minimum cycle cover F is obtained, say, consisting of k cycles ($k \geq 1$). In the example of Figure 2.2, three cycles are generated. If $k > 1$, the subcycles in F can be *patched* into a complete tour.

BnB algorithms use the value of a patching solution as an upper bound by which nodes of the search tree are fathomed.

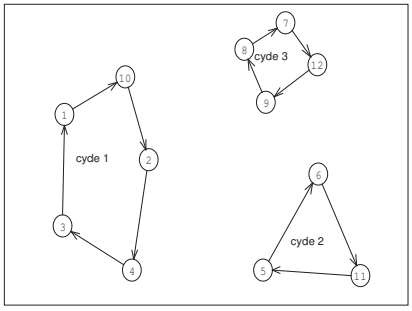

Figure 2.2. Minimum cycle cover

A *patching operation* is the simultaneous deletion of two arcs from a cycle cover and the insertion of two other arcs, such that the number of cycles is reduced by one. In our example, two patching operations are needed for the generation of a complete tour (see Figure 2.3), namely first arcs (2,4), (5,6) are deleted and (2,6) and (5,4) are inserted, and then we delete (12,9) and (2,6) and insert (2,9) and (12,6). The resulting tour is generally feasible but not optimal.

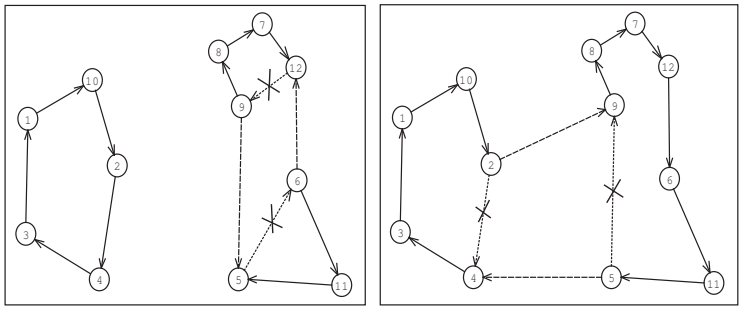

Figure 2.3. Obtaining a tour by means of two patching operations

In Karp (1979), patching is defined as a sequence of $k-1$ patching operations on a cycle cover of k cycles, $k \geq 1$. Recall that even a best possible patching procedure consisting of $k-1$ patching operations does not always yield a shortest complete tour. For example, consider the sparse network in Figure 2.4. The minimum cycle cover consists of the $k = 2$ cycles $(1, 2, 3, 4, 5, 1)$

and $(6, 7, 8, 9, 6)$ with total length 29. The unique shortest complete tour is $(1, 2, 8, 9, 6, 7, 4, 3, 5, 1)$ with length 31. Since four arcs need to be inserted and deleted, this tour cannot be constructed from the cycle cover by means of one patching operation. Different patching procedures are introduced in the literature; see Glover *et al.* (2001); Karp (1979); Karp and Steele (1990); Yeo (1997). These patching procedures are discussed in Section 2.3.

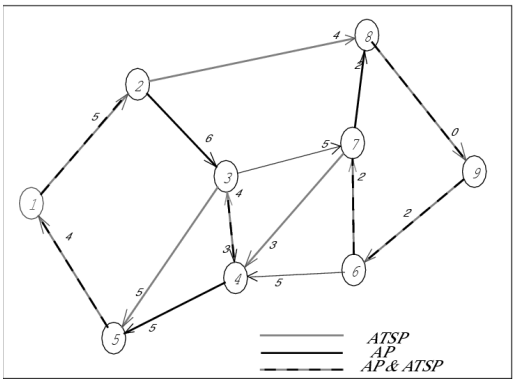

Figure 2.4. Best patching solution is not a shortest tour

Most heuristics for the ATSP apply patching procedures only once, such as to obtain approximations to optimal solutions; see e.g. Glover *et al.* (2001); Gutin and Zverovich (2005); Johnson *et al.* (2002). BnB algorithms apply patching procedures in order to obtain good feasible solutions with which parts of the search tree can be discarded. Any heuristic may be used to generate such solutions, but patching procedures are the most natural choices, since they use the structure of the already constructed minimum cycle cover. If a fixed patching procedure is applied at every node in a BnB algorithm, we call it *iterative patching*.

The currently best BnB algorithms for the ATSP are introduced in Carpaneto *et al.* (1995) and in Miller and Pekny (1991). We call these the CDT algorithm and the MP algorithm, respectively. The CDT algorithm uses the patching procedure from Karp and Steele (1990) at the top node of the search tree. Only if the number of zeroes in the reduced matrix at the top node exceeds a threshold value β, then a subtour-merging procedure is carried out at each node of the search tree.

The subtour-merging procedure constructs first an admissible graph of zero-

cost elements in the reduced matrix and then tries to find a complete tour in the admissible graph. The subtour-merging procedure patches cycles together, but only when a zero-cost patching operation is available. It usually does not return a complete tour. In Carpaneto *et al.* (1995), it is found that if β is set to $2.5n$, the solution times are the shortest, where n is the dimension of the instance.

The MP algorithm applies the Karp-Steele patching procedure, but not at every node of the search tree. Nodes close to the top node are patched more often than nodes deep in the tree. This algorithm also applies a subtour-merging procedure at each node.

The CDT and the MP algorithm both use a best first search (BFS) strategy, which means that a node with the smallest lower bound value is expanded next. BFS is the fastest search strategy, but requires exponential memory space. As a consequence, BFS algorithms are generally restricted to small or easily solvable problems (Zhang, 1993). In depth first search (DFS), the most recently generated subproblem is solved first, and it requires polynomial memory space. This makes it suitable for solving large and difficult instances. However, the search trees and solution times of DFS algorithms are usually large.

Miller and Pekny (1991) report that iterative patching is too time-consuming. This may be true for BFS algorithms, but our algorithms use DFS. DFS algorithms search through deep nodes of the search tree even at an early stage; lower bounds of such nodes are generally high. A tight upper bound obtained early enables the algorithm to discard a large fraction of these nodes. Therefore, a DFS algorithm is more likely to benefit from a good upper bounding procedure, such as iterative patching, than a BFS algorithm.

The computational experiments in Section 2.4 compare the search tree sizes and the running times of BnB algorithms that apply iterative patching with a DFS implementation of the CDT algorithm. We apply four patching procedures, namely the ones discussed in Glover *et al.* (2001). The main questions that we answer on iterative patching in this paper are as follows. Is iterative patching effective for DFS algorithms? Is it true that if a patching procedure returns on average shorter tours than some other one, then, again on average, the search tree sizes are smaller and the running times are shorter? Hence, does better patching lead to the smaller search trees and shorter running times?

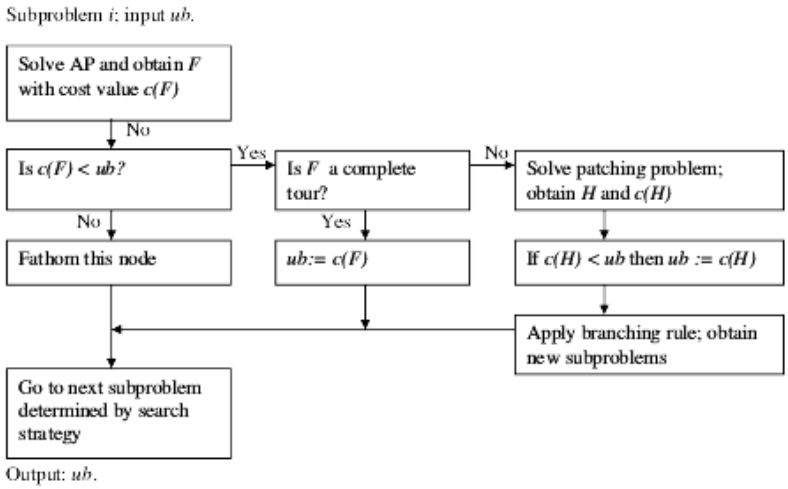

Figure 2.5. Flowchart of a subproblem of a BnB algorithm with iterative patching

2.2 The quality of patching procedures

Let $G(V, A)$ be a graph with vertex set V and arc set A. A minimum cycle cover $F \subset A$ can be determined in $O(n^3)$ time by means of the Hungarian algorithm; see for example Jonker and Volgenant (1986). The speed of the Hungarian algorithm can be increased in successor nodes j to $O(n^2)$ by starting from the optimal solution in the parent node, i.e., the node in which subproblem j is generated; see for example Fischetti *et al.* Fischetti *et al.* (2002).

Patching procedures delete pairs of arcs from F and insert pairs of arcs from $A \setminus F$ in such a way that a Hamiltonian cycle $H \subset A$ is obtained. The patching cost of any patching procedure P is then denoted by $c_P(F)$ and defined as

$$c_P(F) = \sum_{a \in H \setminus F} c(a) - \sum_{b \in F \setminus H} c(b), \tag{2.2.1}$$

where $c(a)$ denotes the cost of arc $a \in A$. The first term of (2.2.1) indicates the cost of the new arcs introduced by P, and the second term represents the cost of the arcs removed from the cycle cover. For any subset $Q \subset A$, $c(Q)$ denotes the sum of the cost of the arcs in Q.

Let $F_j \subset A$ denote a minimum cycle cover at node j of the BnB search tree

in progress. By $BnB(Br, S, UBS)$ we denote a BnB algorithm for the ATSP that applies branching rule Br, search strategy S, and upper bounding strategy UBS. A branching rule Br partitions the current feasible regions into subsets. We consider branching rules that only depend on the current minimum cycle cover. The search strategy S in this paper is DFS. The upper bounding strategy UBS consists of two components: the first component prescribes at which nodes an upper bounding procedure should be applied, and the second component specifies the upper bounding procedure to be used. Clearly, iterative patching is an upper bounding strategy, where a tour is generated at every node of the search tree by means of a fixed patching procedure. If no confusion is likely, we simply write $BnB(UBS)$, since S and Br are fixed in this study.

Note that, in case of DFS, the order of node expansion is independent of the bounds used at each subproblem. For instance, if both algorithms $BnB(P_1)$ and $BnB(P_2)$ explore two subproblems S_1 and S_2, and $BnB(P_1)$ explores S_1 before S_2, then $BnB(P_2)$ will explore S_1 before S_2 as well.

Let $ub_j(UBS)$ be the current *upper bound*, i.e. the shortest complete tour obtained until node j using upper bounding strategy UBS. Recall that, when the UBS is iterative patching, we obtain at each node of the search tree a complete tour, i.e. a candidate for the value of $ub_j(UBS)$.

Node k is called a *successor* of j in a search tree if j is an intermediate node of the shortest path between k and the top node of the search tree; we use the notation $k \propto j$. Since the feasible region of the AP at node k is a subset of the feasible region of the AP at node j, we have of course that $c(F_k) \geq c(F_j)$ if $k \propto j$; see e.g. Zhang (1993).

In the case of iterative patching, one may expect that if patching costs are low, then upper bounds are tighter and a larger number of subproblems can be fathomed. Theorem 2.2.1 formalizes this assertion: if for each instance patching procedure P_1 is cheaper than patching procedure P_2, then the search tree of $BnB(P_1)$ will be smaller than the search tree of $BnB(P_2)$.

For any iterative patching procedure P, let $BnB(P)$ be the algorithm that uses P iteratively. Define $\#BnB(P)$ as the size of the solution tree of $BnB(P)$, i.e. the number of nodes in this tree. We assume in Theorem 2.2.1 that $BnB(P_1)$ and $BnB(P_2)$ use the same AP-solver implementation, since the choice of another AP solver may result another initial minimum cycle cover. The cycle cover

is the starting point of the patching procedure; if the minimum cycle covers are different, then the resulting patching solutions and their costs may differ, even though the same patching procedure is applied.

Theorem 2.2.1. *Let $\mathcal{F}$ be the set of minimum cycle covers of a given instance of the ATSP, and let P_1 and P_2 be two patching procedures such that their respective patching costs satisfy $c_1(F) \leq c_2(F)$ for each $F \in \mathcal{F}$. It then follows that $\#BnB(P_1) \leq \#BnB(P_2)$.*

Proof. For any given instance of the ATSP, let $T(Br)$ be the complete search tree based only on branching rule Br, i.e. the search tree in which all possible solutions are enumerated. Usual BnB procedures apply the following pruning operations:

1. If at a certain node of $T(Br)$ F is a complete tour, then all successor nodes are deleted from $T(Br)$.

2. If at a certain node of $T(Br)$, say j, it holds that $c(F_j) \geq ub_j(P)$, then this node and all its successors are fathomed.

For any patching procedure P, $BnB(P)$ deletes nodes from the complete search tree $T(Br)$ until the usual BnB tree remains, which we denote by $T(P)$. Clearly, pruning operation (1) is independent of the patching procedure used, since the AP solver implementation is taken fixed. Actually, at each node the same minimum cycle cover is found.

We now show that $T(P_1) \subseteq T(P_2)$ by showing that if node j is fathomed under P_2, then it is also fathomed under P_1. This is the case, if for each node j, it holds that $c(F_j) \geq ub_j(P_2) \Longrightarrow c(F_j) \geq ub_j(P_1)$. So we need to show that $ub_j(P_1) \leq ub_j(P_2)$ for each node j on the path obtained by the search strategy S. Thus, $BnB(P_2)$ is only able to discard nodes if $BnB(P_1)$ discards them, which implies that $\#BnB(P_1) \leq \#BnB(P_2)$.

Obviously, for the first node $j = 0$, it holds that $ub_0(P_1) \leq ub_0(P_2)$. Now assume that $ub_j(P_1) \leq ub_j(P_2)$ at node j. Let k be the next unsolved subproblem after node j according to the search strategy S. We show that $ub_k(P_1) \leq ub_k(P_2)$. Let $H_P(F)$ be the patching solution of procedure P given minimum cycle cover F.

After solving the AP at node j, both algorithms compare $c(F_j)$ with their current upper bounds. Three scenarios are possible:

1. If $ub_j(P_1) \leq ub_j(P_2) \leq c(F_j)$, then both algorithms fathom node j and both procedures proceed to node k. Clearly, $ub_k(P_1) = ub_j(P_1) \leq ub_j(P_2) = ub_k(P_2)$.

2. If $c(F_j) < ub_j(P_1) \leq ub_j(P_2)$, then both algorithms execute patching at node j. Since $c_1(F_j) \leq c_2(F_j)$, it follows that $c(H_1(F_j)) = c(F_j) + c_1(F_j) \leq c(F_j) + c_2(F_j) = c(H_2(F_j))$. Since $ub_k(P_i) = \min\{ub_j(P_i), c(H_i(F_j))\}$ for $i = 1, 2$, we have that $ub_k(P_1) \leq ub_k(P_2)$.

3. If $ub_j(P_1) \leq c(F_j) < ub_j(P_2)$, then $BnB(P_1)$ fathoms node j, and $ub_k(P_1) := ub_j(P_1)$. $BnB(P_2)$ solves an additional patching problem at node j and possibly at the successor nodes of j. Let q be the successor node of j in which the best patching solution is obtained, i.e. $q = \arg\min_l\{c(H_2(F_l)); l \propto j, l = j\}$. After searching through all successors of j, or after discarding them, $BnB(P_2)$ arrives at node k with $ub_k(P_2) \geq \min\{ub_j(P_2), c(H_2(F_q))\}$. Clearly, $ub_k(P_1) = ub_j(P_1)$. Furthermore, it holds that $ub_j(P_2) \geq ub_j(P_1) = ub_k(P_1)$, and that $c(H_2(F_q)) \geq c(F_q) \geq c(F_j) \geq ub_j(P_1) = ub_k(P_1)$. Hence, $ub_k(P_2) \geq ub_k(P_1)$.

Hence, for all nodes j on the path according to S through $T(Br)$, we have that $ub_j(P_1) \leq ub_j(P_2)$. Therefore, $\#BnB(P_1) \leq \#BnB(P_2)$. □

Theorem 2.2.1 can be extended to upper bounding strategies UBS for which the upper bound generated at node j is at least $c(F_j)$. In that case, upper bounds are only obtained at nodes at which a complete tour is constructed; elsewhere, the patching costs are infinite. For example, consider a BnB algorithm $BnB(P; ni)$ that applies patching procedure P not iteratively. It follows from Theorem 2.2.1 that its search tree is always at least the size of the search tree of the algorithm $BnB(P)$ that applies P iteratively.

In general, there are few iterative patching procedures that always return better patching solutions than some other one. Therefore, it makes more sense to consider the average performance of patching procedures. To this end, we conduct computational experiments in Section 2.4.

The most important measure of the quality of algorithms are solution times. Actually, high quality patching solutions may lead to long solution times of subproblems. So usually, a trade-off is made between the quality of the patching and time invested in patching. For instance, if patching procedure P is only applied at the top node, the search tree is larger than the tree with iterative patching procedure P. However, the average solution time at the nodes is smaller. In Section 2.4, solution times are taken into account more explicitly.

The following observation allows to increase the speed of iterative patching without losing quality. Recall that, if a cycle cover F consists of k cycles, patching is a sequence of $k-1$ patching operations. Call the cycle cover after the i-th patching operation F_i, and denote its cost by $c(F_i)$, $i = 1, \ldots, k-1$. If $c(F_i)$ exceeds the cost of the current best solution ub, the patching procedure will certainly not lead to a better solution, since the cost of each patching operation is nonnegative. Hence, we can abort the patching after i steps and save running time.

2.3 Patching Procedures

We now compare the performance of four iterative patching procedures based on the four most well-known patching algorithms. We start with a short description of these four patching procedures. All these procedures have a worst-case time complexity of $O(n^3)$, see Glover *et al.* Glover *et al.* (2001).

Karp-Steele patching (KSP) was introduced in Karp and Steele (1990). Starting with the minimum cycle cover F, KSP patches the two longest subcycles successively by using a cheapest patching operation. In our example, KSP patches cycles 1 and 3 by deleting (10,2) and (9,8), and adding (10,8) and (9,2); see Figure 6. The new cycle is then patched with cycle 2 by removing (12,9) and (5,6), and inserting (5,9) and (12,6).

Modified Karp-Steele patching (MKS), also called *Greedy Karp-Steele patching*, see Glover *et al.* (2001), performs the cheapest patching operation among all pairs of cycles in the current cycle cover. The patching costs are then updated and the procedure is repeated until a complete tour is obtained. Since it compares in general more patching operations than KSP, MKS is more time-consuming. In our example, MKS joins cycles 2 and 3 by deleting arcs (5,6) and (12,9), and

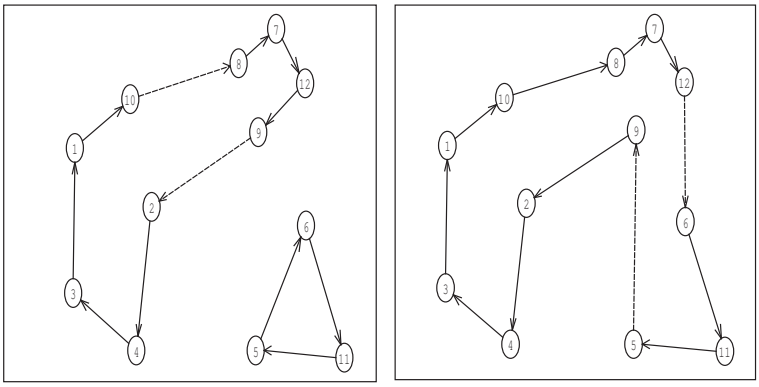

Figure 2.6. Modified Karp-Steele patching in action

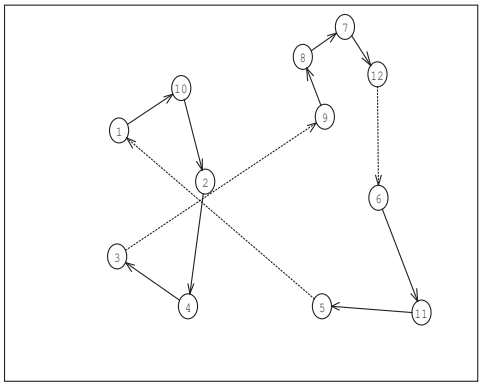

Figure 2.7. RPC patching solution

inserting (5,9) and (12,6). Cycle 1 is included by inserting (2,9) and (5,4) and removing (2,4) and (5,9); see Figure 2.6.

Recursive Path Contraction (RPC) was introduced in Yeo (1997). From all, say k, cycles a most expensive arc is deleted and the remaining paths are contracted, so transformed into single nodes. On these k nodes an AP is solved. So every contracted path is connected to another contracted path. The procedure is carried out recursively until one cycle is obtained. The calculations of Section 2.4 use the implementation from Glover *et al.* (2001). In our example, the most expensive arc from every cycle is deleted, namely (3,1), (5,6) and (12,9). The end nodes 3, 5 and, 12 are assigned to nodes 9, 1, and 6, respectively. Finally, the tour depicted in Figure 2.7 is obtained.

Contract-or-Patch (COP) is a two-stage procedure consisting of RPC in the first stage and, either MKS or KSP in the second stage; see Glover *et al.* (2001) and Gutin and Zverovich (2005). All cycles with length less than a user-defined threshold value t are patched using RPC. In Gutin and Zverovich (2005), it is shown that the threshold value $t = 5$ is the most robust choice for different types of instances. Given the cycle cover from Figure 2, cycles 2 and 3 are patched using the RPC procedure. The long cycles in the current cycle cover are patched with either KSP or MKS. In Section 2.4, the faster procedure KSP is selected, since in Johnson *et al.* (2002) it is asserted that there is no significant difference in the patching cost of COP using either KSP or MKS.

2.4 Computational experiments

In this section, we compare both the tree sizes and the running times of the algorithms presented in Table 2.1. Recall that the size of a BnB tree is the number of subproblems solved before the first optimal solution is determined, i.e. the number of nodes visited on the path followed through $T(Br)$ according to search strategy S. The results of iterative patching procedures are compared with the results of the DFS implementation of the CDT algorithm. The DFS implementation is of practical use, because it solves ATSPLIB and symmetric instances which a BFS approach cannot solve; see for example Carpaneto *et al.* (1995) and Miller and Pekny (1991).

Table 2.1. Patching strategies tested

Name	Patching strategy
$BnB(KSP)$	Iterative KSP
$BnB(MKS)$	Iterative MKS
$BnB(RPC)$	Iterative RPC
$BnB(COP)$	Iterative COP
$BnB(CDT)$	DFS implementation of CDT algorithm

The experiments are performed on a Pentium 4 computer with speed 2 GHZ and 256 MB RAM under Windows 2000. The programming language is C and the compiler is GNU with speed -o2. Our branching rule branches by a largest cost arc in the shortest subcycle of a minimum cycle cover. In a forthcoming study we will apply tolerance-based branching rules, where branching is per-

formed on an arc with the smallest tolerance value (the amount at which the cost can be changed without changing the solution at hand). The iterative patching procedures are tested for the following types of instances:

1. Asymmetric TSPLIB instances (see Reinelt (1995));
2. Randomly generated instances with varying degree of symmetry;
3. Randomly generated instances with varying degree of sparsity;
4. Random instances with a large number of different intercity distances;
5. Almost symmetric Buriol instances (see Buriol *et al.* (2004)).

From all asymmetric TSPLIB instances we have selected 16 instances that are solvable within reasonable time limits. The random instances have degree of symmetry 0, 0.33, 0.66, and 1, where the *degree of symmetry* is defined as the fraction of off-diagonal entries in the cost matrix $\{c_{ij}\}$ that satisfy $c_{ij} = c_{ji}$. The third class of instances consists of instances with varying *degree of sparsity*, being defined as the fraction of the total possible number of arcs that are missing. We study instances with degree of sparsity of 0, 0.25, 0.5, and 0.75.

If both the degree of symmetry and degree of sparsity of an instance is unrestricted, we call such an instance *usual random*. The usual random instances have problem size 60, 70, 80, 100, 200, 300, 400, and 500. Random instances with degree of symmetry larger than 0 have problem size 60, 70, and 80. Only these samples of (quasi-)symmetric instances are considered, since computation times for larger symmetric instances tend to be extremely long. The instances with varying degree of sparsity have problem size 100, 200, and 400. The arc costs are drawn from a discrete uniform distribution supported on $\{1, 2, ..., 10^4\}$; for each problem set and for all problem sizes, 10 instances are generated. In comparison with other studies, namely Carpaneto *et al.* (1995) and Miller and Pekny (1991), our random instances are relatively small, whereas our symmetric instances are relatively large. For example, the MP algorithm by Miller and Pekny (1991) solves random instances of size 500000, but solves symmetric instances of size less than 30 only.

In addition to the usual random instances, we generate random instances with a large number of different intercity distances. The reason for considering

these instances is given in Zhang (2003, 2004), where it is shown that if the number of different intercity distances exceeds a threshold value, the instance becomes relatively hard to solve. Suppose the arc costs of an instance are uniformly distributed on $\{1, \ldots, R\}$, where R is the range of the distribution. It is shown in Zhang (2003) that, as the range increases, the number of intercity distances increases as well. Moreover, uniform random instances are hard to solve if the range R is at least n^2; see Zhang (2004). This result implies that our randomly generated instances with size larger than 100 are relatively easy to solve. Therefore, we use additional 'hard' random instances with arc costs drawn from a uniform distribution supported on $\{1, ..., 10^5\}$ for n = 200, 300, and on $\{1, \ldots, 2 \times 10^5\}$ for n = 400.

Finally, the almost symmetric Buriol instances are asymmetric TSP instances which are derived from symmetric instances from the TSPLIB. They are constructed as follows. Let σ be the average of all distances of the original symmetric TSPLIB instance, and let k' be a user-defined percentage. Then each entry in the lower diagonal of the cost matrix C of the symmetric instance is increased by $k\sigma$, where k is randomly drawn from the uniform distribution supported on $\{0, \ldots, k'\}$. However, the costs of edges belonging to a chosen optimal tour of the original symmetric instance are not modified. Note that the smaller the value of k', the higher the degree of symmetry of the instances generated. We construct eight instances for which $k' = 5$, and twelve for which $k' = 50$, respectively, using the instance generator from Buriol *et al.* (2004). These are instances which are solvable within reasonable time limits.

The average *size of the search tree* of the algorithms is shown in Table 2. In order to make the results more comparable, we have used *normalized* results, i.e., we have fixed the results of $BnB(CDT)$ at 100. The number '50.65' in the MKS-column means that the $BnB(MKS)$ generates on average about half the number of subproblems of $BnB(CDT)$ for instances with degree of symmetry 0.33.

Table 2.2 shows that, except for the RPC procedure, iterative patching leads to smaller search trees. The search tree reductions of iterative patching are large for usual random and sparse instances; the sizes of the trees of $BnB(KSP)$, $BnB(MKS)$ and $BnB(COP)$ are half the size of the search tree of $BnB(CDT)$. The reductions of iterative patching are smaller for symmetric and

Table 2.2. Normalized size of search tree for usual BnB (CDT = 100)

	CDT	KSP	MKS	RPC	COP
ATSPLIB	100.00	95.03	94.27	101.40	95.37
Buriol, $k' = 5$	100.00	98.37	97.39	98.97	98.01
Buriol, $k' = 50$	100.00	78.64	39.95	102.95	95.70
Usual random	100.00	47.27	43.97	129.98	47.27
Random, large range	100.00	48.99	48.99	167.83	48.99
Degree of symmetry 0.33	100.00	50.81	50.65	106.75	51.16
Degree of symmetry 0.66	100.00	74.52	73.66	101.45	75.44
Full symmetry	100.00	99.79	99.77	99.97	99.80
Degree of sparsity 0.25	100.00	51.66	51.20	113.26	51.66
Degree of sparsity 0.50	100.00	56.13	56.13	126.68	56.13
Degree of sparsity 0.75	100.00	56.43	56.35	129.98	56.43

ATSPLIB instances. $BnB(MKS)$ ad also $BnB(KSP)$ have relatively large search tree reductions for the almost symmetric Buriol instances with $k' = 50$, but $BnB(COP)$ is performing considerably worse. On average, the search trees generated by $BnB(MKS)$ are the smallest, whereas $BnB(RPC)$ only generates reasonably small search trees for symmetric instances.

Table 2.3. Normalized running times

	CDT	KSP	MKS	RPC	COP
ATSPLIB	100.00	114.81	139.56	114.44	116.01
Buriol, $k' = 5$	100.00	111.31	158.72	130.00	125.56
Buriol, $k' = 50$	100.00	91.25	57.53	116.91	112.69
Usual random	100.00	55.81	60.24	140.44	54.45
Random, large range	100.00	61.05	81.07	191.03	64.88
Degree of symmetry 0.33	100.00	72.22	72.22	170.83	55.56
Degree of symmetry 0.66	100.00	93.33	103.70	132.22	85.00
Full symmetry	100.00	108.24	126.76	114.98	111.54
Degree of sparsity 0.25	100.00	62.64	73.57	125.13	62.33
Degree of sparsity 0.50	100.00	69.05	90.16	144.79	77.44
Degree of sparsity 0.75	100.00	73.79	85.33	153.29	73.88

In Table 2.3, we present the normalized *running times*. For usual random and sparse instances, iterative patching is clearly more effective; the search tree reduction outweighs the time invested in patching at nodes. Although $BnB(MKS)$ often requires the smallest search trees, $BnB(COP)$ and $BnB(KSP)$ mostly

display smaller running times. This indicates that the speed of solving patching problems is relevant. Solution times of iterative patching are longer for instances from the ATSPLIB and for symmetric instances than of $BnB(CDT)$, although in both cases the differences are small.

The following tables show the absolute search tree sizes and solution times in more detail. For most ATSPLIB instances, the search tree reductions of iterative patching are minor, and the solution times increase; see Table 3.5. For the usual random instances, the iterative patching procedures $BnB(KSP)$, $BnB(MKS)$, and $BnB(COP)$ have clearly smaller search tree sizes and solution times than $BnB(CDT)$; see Table 2.5. These benefits appear to be independent of the instance size. Finally, Table 2.6 and Table 2.7 present the absolute tree sizes and solution times of sparse and symmetric instances.

The results for the almost symmetric instances from Buriol *et al.* (2004) are presented in Table 2.8 and 2.9. They indicate that patching does not lead to shorter solution times for most instances with $k' = 5$, but it becomes worthwhile if the deviation k' is increased to 50.

Table 5.3 presents the results for the random instances with a large number of different intercity distances. We obtain similar results as for the usual random instances: the solution times of $BnB(KSP)$, $BnB(MKS)$, and $BnB(COP)$ are clearly lower than those of $BnB(CDT)$ and $BnB(RPC)$. So iterative patching is not sensitive to changes in the range of the uniform distribution.

Symmetric and ATSPLIB instances can be considered 'hard', i.e., even small instances have large search trees and running times. For these instances, cycle covers often consist of many short cycles. Hence, tours obtained by patching are long, and only minor parts of the search tree can be discarded, so the small reductions of the search tree do not compensate for the time invested in patching at each node. This explains the special behavior of symmetric and ATSPLIB instances. The same holds for the almost symmetric Buriol instances.

Table 6 and Figure 2.8 show that, as the degree of symmetry increases, the search trees of $BnB(CDT)$ and $BnB(RPC)$ converge to the size of the other trees. Hence, applying iterative patching makes no sense for symmetric instances. On the other hand, the degree of sparsity does not influence the relative search tree sizes of the algorithms; see Figure 2.8. So sparsity does not influence the usefulness of iterative patching.

Table 2.4. Search tree sizes and solution times (seconds) of ATSPLIB instances

	CDT		KSP		MKS		RPC		COP	
Instance	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
ft53	21189	2.20	20111	2.31	20111	2.64	21189	2.36	20111	2.42
ft70	26025	3.57	25831	3.85	25831	4.40	26025	4.01	25831	4.07
ftv33	7455	0.16	7065	0.22	7061	0.27	7307	0.22	7065	0.22
ftv35	7305	0.16	6945	0.16	6939	0.22	8267	0.22	6951	0.22
ftv38	7325	0.22	6195	0.22	6195	0.27	10101	0.38	6195	0.16
ftv44	3753	0.11	619	0.01	619	0.05	3753	0.16	3083	0.16
ftv47	29539	1.10	29025	1.26	29017	1.76	29539	1.32	29031	1.37
ftv55	114403	4.73	92447	4.51	92447	5.82	114785	5.44	103839	5.55
ftv64	252755	11.87	43441	3.19	43441	4.18	252755	15.93	43441	3.52
ftv70	326827	23.41	253873	24.95	206195	27.36	410545	35.60	261199	24.73
ftv170	1796439	1073.63	1796149	1300.88	1796159	1614.56	1796459	1198.96	1796149	1276.87
rbg323	3	0.05	3	0.05	1	0.05	9	0.05	3	0.01
rbg358	3	0.05	3	0.05	1	0.16	7	0.11	5	0.05
rbg403	3	0.05	3	0.05	1	0.11	7	0.11	3	0.05
rbg443	3	0.05	3	0.05	1	0.11	3	0.11	3	0.05
br17	3674829	16.59	3674829	24.23	3674829	32.69	3674829	24.51	3674829	24.40

Table 2.5. Search tree sizes and solution times (seconds) of usual random instances

	CDT		KSP		MKS		RPC		COP	
n	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
60	6508	0.60	3808	0.38	3808	0.44	12880	1.10	3808	0.33
70	10828	1.21	4528	0.44	4528	0.71	18522	2.14	4528	0.55
80	21834	2.75	9014	1.26	8622	1.48	27822	4.1	9014	1.26
100	13454	2.42	9002	1.92	6814	1.81	17424	3.73	9002	1.98
200	138522	114.	36390	33.	36390	40.	172054	151.	36390	33.
300	412930	798.	178498	481.	178498	551.	500100	1081.	178498	424.
400	525088	2142.	284994	1410.	284982	1746.	640440	2825.	284994	1349.
500	951188	6428.	434576	3687.	432000	5284.	1456440	10868.	434576	3889.

Table 2.6. Search tree sizes of symmetric and sparse instances

	CDT	KSP	MKS	RPC	COP
Instance	Size	Size	Size	Size	Size
Degree of symmetry 0.33	122520	58878	58724	129914	59458
Degree of symmetry 0.66	259626	202894	200630	264444	204470
Full symmetry	114984046	114912026	114908592	109843207	114915850
Degree of sparsity 0.25	637872	362188	354610	732500	362188
Degree of sparsity 0.50	653016	368736	368736	801526	368736
Degree of sparsity 0.75	704832	386468	386392	883026	386468

Table 2.7. Solution times (seconds) of symmetric and sparse instances

	CDT	KSP	MKS	RPC	COP
Instance	Time	Time	Time	Time	Time
Degree of symmetry 0.33	13	8	8	19	7
Degree of symmetry 0.66	33	33	38	45	32
Full symmetry	17584	19182	22521	19271	19972
Degree of sparsity 0.25	1935	1451	1801	2386	1434
Degree of sparsity 0.50	1797	1341	1746	2350	1345
Degree of sparsity 0.75	1998	1467	1909	2857	1472

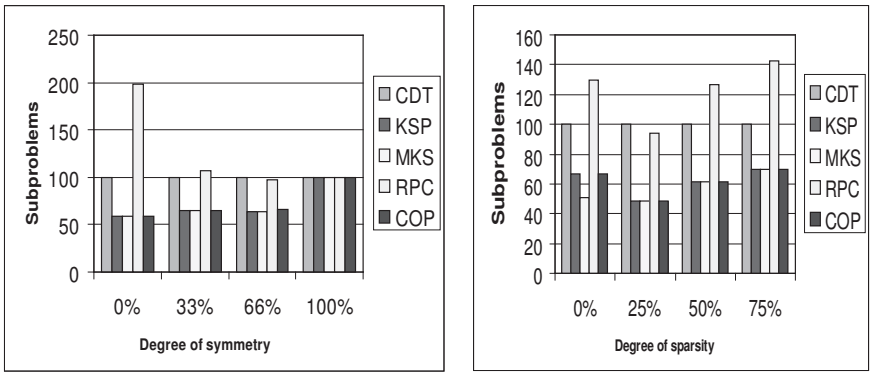

Figure 2.8. Normalized search tree sizes of instances with varying degree of symmetry (n = 60) and sparsity (n = 100), CDT = 100

The major drawback of BnB algorithms is their time consumption: it may take very long before an optimal solution is obtained. When the BnB process is terminated and the best solution so far is taken, the procedure is called Truncated Branch-and-Bound (TBnB); see Zhang (2000). Usually, the TBnB algorithm is terminated if a predefined number of nodes in the search tree is expanded. Currently, TBnB uses KSP only. An interesting question for future research is whether TBnB can be improved by including other iterative patching procedures.

Premature termination of BnB is effective if good solutions are found at an early stage of the BnB process, and a large portion of the time is spent on proving optimality. In Table 2.4, we present the solution quality for difficult practical instances from Buriol *et al.* (2004) with $k' = 5$. The solution quality reported in the table is the relative gap between the optimal solution of the instance and the best solution found by the BnB algorithm after a fixed number of subproblems. The results indicate that, even after solving a large number of subproblems, the BnB solutions are still far from optimal. For example, solving problem instance pr76 takes about 12 hours; about 50% of the time is spent on finding an optimal solution. So when our BnB methods are terminated in an early stage, the resulting solutions are not competitive with meta-heuristic solutions for these instances. Table 2.4 also shows that iterative KSP finds good solutions in an early stage, whereas the top node patching algorithm has to solve a very large number of subproblems before achieving the same solution quality.

All BnB algorithms presented in this paper are able to solve small instances in more or less the same amount of time as most meta-heuristics do. However, our computational experiments indicate that these BnB methods are not able to find optimal solutions within reasonable time limits for large, almost symmetric, Buriol instances with $k' = 5$, or for fully symmetric instances. On the other hand, meta-heuristics generate solutions to these instances within a few percents from the optimal solution value in fractions of seconds; see for example the survey paper Buriol *et al.* (2004). The errors of meta-heuristics are in the range 0 to 0.44% for ATSPLIB instances, and below 0.6% for the almost symmetric Buriol instances. However, these results do not imply that meta-heuristics are always preferable over BnB methods. Although the errors appear pretty small, a solution with 0.5% higher cost than optimal may be very costly in practice.

A new line of research is combining the force of BnB and meta-heuristics,

in particular *memetic algorithms* (Moscato and Cotta, 2003). In evolutionary algorithms the elements, paths in case of the ATSP, inherited from the parents are recombined, but memetic algorithms use optimization methods to construct good feasible solutions. Similar to BnB, the question is how much time should be spent on the optimization of each agent's tour. Buriol *et al.* (2004) use a type of patching algorithm for the optimization, and the memetic algorithm by Cotta and Troya constructs such tours by means of a Branch-and-Bound subroutine (Cotta and Troya, 2003).

The results indicate that it is worthwhile to invest time in finding good upper bounds in BnB algorithms with a DFS strategy. The method for finding a good upper bound need not be patching, and need not be iterative as well. Another interesting question is whether it is worthwhile to invest more time in finding a tight upper bound at the top node of the BnB search tree. For that purpose, a meta-heuristic can be applied instead of patching. Klau *et al.* (2004) apply a memetic algorithm initially for preprocessing and to obtain a good starting solution for the Biobjective Flowshop Scheduling Problem. The same strategy is followed by Basseur *et al.* for the Prize-Collecting Steiner Tree (Basseur *et al.*, 2004). A hybrid application of meta-heuristics and Branch and Bound may form fertile area of future research.

In Glover *et al.* (2001), the performance of patching heuristics on solution quality is studied. The results show that MKS returns the best patching solutions for ATSPLIB instances, and COP for random instances, both symmetric and asymmetric. In Table 2.4, the solution quality results from Glover *et al.* (2001) are compared with our search tree sizes. The results show that the ordering with respect to solution quality of patching procedures differs from the ordering with respect to search tree sizes of the corresponding iterative patching procedure. This phenomenon may be caused by the following effect. Recall that, when iterative patching is applied, patching solutions are constructed at each node of the search tree. It may be misleading to take into consideration the patching quality only at the top node of the search tree, and expect that for all nodes in the search tree on average the same quality holds. Actually, it is more likely that good upper bounds are found deep in the search tree and that the average patching solution quality deep into the tree differs from the average top node patching quality. In fact, top node cycle covers may consist of many short cycles, whereas subcycles

tend to become longer as the BnB algorithm proceeds deeper into the search tree, because our branching rule attempts to break short cycles. This may explain the differences in the orderings according to the average patching quality and to the average search tree size of the iterative patching procedures.

Consider for example the iterative patching procedures RPC and COP. $BnB(RPC)$ needs long running times and large search trees for random instances, because RPC deletes an arc from every cycle without calculating patching costs. Therefore, if cycles are long, bad patching operations are likely. COP, on the other hand, patches long cycles carefully, leading to smaller search trees.

2.5 Conclusion

We studied the performance of four iterative patching procedures, being fixed patching procedures at every node of the search tree, which we compared with the performance of a depth first search implementation of the CDT algorithm by Carpaneto *et al.* (1995). Our performance measures are the size of the search tree and the running times of the algorithms. Clearly, there is a trade-off between the quality of patching, leading to smaller search trees, and the speed of solving each patching problem. We conclude with an answer to the main questions.

Is it worthwhile to use iterative patching procedures? At least, search trees are always smaller. However, only for 'practical' instances the solution times are shorter when $BnB(CDT)$ is applied. A side effect of iterative patching is that, if calculations are finished prematurely, a satisfactory solution is often at hand; see Zhang (1993). An interesting direction of future research is to study iterative patching procedures in Truncated BnB algorithms and other meta-heuristics. Another interesting direction is to find intermediate strategies between full iterative patching and top node patching. To this end, the nodes of the search tree must be identified at which a good patching solution can be expected.

Which iterative patching procedure is the most efficient one? On the whole, the algorithm using MKS generates the smallest solution trees, and our COP and KSP implementations achieve the best solution times. The most important performance criterion is usually the solution time. However, if the memory of the computer is the restrictive factor, then it also becomes important to keep the search trees small.

Table 2.8. Running times and search tree sizes for almost symmetric Buriol instances, $k' = 5$

	CDT		KSP		MKS		RPC		COP	
Instance	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
ulysses16	10607	0.05	10165	0.05	10139	0.11	10877	0.16	10123	0.11
ulysses22	863039	8.41	857027	13.08	818327	11.65	863039	12.31	832311	10.33
bayg29	12443	0.27	6555	0.16	5563	0.16	12443	0.44	6555	0.22
bays29	17283	0.33	13931	0.33	12355	0.33	16471	0.49	14687	0.33
eil51	1197185	57.64	1193015	64.89	1193015	89.78	1197185	79.34	1196191	72.25
fri26	12891	0.22	12725	0.22	9187	0.27	12891	0.33	12759	0.27
gr24	2311	0.05	1329	0.05	1329	0.00	2311	0.05	2299	0.05
gr48	3400347	155.55	3331593	168.90	3322435	250.88	3344327	196.15	3331611	195.82

Table 2.9. Running times and search tree sizes for almost symmetric Buriol instances, $k' = 50$

	CDT		KSP		MKS		RPC		COP	
Instance	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
ulysses16	2553	0.00	2481	0.05	2481	0.00	12025	0.11	7655	0.05
ulysses22	94831	0.88	35577	0.44	17711	0.27	99963	1.21	89035	1.10
att48	181943	7.53	147201	7.42	5967	0.38	186547	9.45	147201	7.25
bayg29	217	0.00	177	0.00	13	0.05	217	0.00	211	0.00
bays29	2289	0.11	323	0.00	223	0.00	2613	0.05	521	0.05
eil51	21689	1.10	16869	0.99	951	0.05	26399	1.65	20497	1.26
fri26	2495	0.05	2481	0.05	2481	0.05	2495	0.05	2481	0.05
gr24	141	0.00	49	0.00	49	0.00	141	0.00	141	0.00
gr48	5213	0.33	4503	0.33	2405	0.16	5213	0.33	5177	0.38
pr76	1659	0.27	1643	0.27	415	0.05	17229	2.64	5293	0.88
eil76	567	0.05	243	0.05	239	0.00	567	0.11	243	0.00
gr96	1038095	262.91	851473	239.73	507019	156.15	1038095	303.85	1015109	296.87

Table 2.10. Search tree sizes and solution times (seconds) of random instances with a large number of different intercity distances

	CDT		KSP		MKS		RPC		COP	
n	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
200	153792	146.32	42358	43.85	42358	52.36	42358	54.73	185116	174.40
300	363654	824.01	184752	470.33	184752	608.13	184752	600.93	454114	1065.38
400	589564	2640.22	271034	1394.23	271034	1815.71	271034	1739.07	1035296	4775.33

Table 2.11. Solution quality after number of subproblems solved for almost symmetric instances with $k' = 5$

	$BnB(KSP)$			$BnB(CDT)$		
Subproblems	1000	10000	100000	1000	10000	100000
pr76	9.78%	9.78%	9.78%	20.28%	19.22%	16.08%
eil76	7.25%	7.25%	7.06%	21.38%	19.52%	13.57%
gr96	7.02%	7.02%	7.02%	14.47%	13.77%	11.55%
kroD100	13.17%	13.17%	13.17%	39.34%	38.85%	37.11%
rd100	13.50%	13.50%	13.50%	40.09%	39.12%	30.78%
lin105	10.60%	10.60%	10.60%	26.64%	25.88%	24.14%
ch130	15.04%	15.04%	15.04%	32.09%	31.93%	28.82%
ch150	16.56%	16.56%	16.56%	36.57%	36.57%	36.17%
brg180	16.77%	16.41%	14.46%	18.62%	18.51%	15.03%
Average time (sec.)	0.48	4.05	35.11	1.30	2.78	28.75

Table 2.12. Ordering of the top node solution quality and the number of iterations

	Average relative excess over AP lower bound		Normalized search tree size (CDT = 100)	
ATSPLIB	MKS	3.36%	MKS	86.15
	KSP	4.29%	KSP	87.99
	COP	4.77%	COP	88.81
	RPC	18.02%	RPC	103.38
Usual random	COP	1.88%	MKS	43.97
	MKS	3.36%	COP	47.27
	KSP	3.11%	KSP	47.27
	RPC	106.65%	RPC	129.98
Full symmetry	COP	79.87%	MKS	99.77
	RPC	183.57%	KSP	99.79
	MKS	586.92%	COP	99.80
	KSP	744.22%	RPC	99.97

Chapter 3

Tolerance-based Branch and Bound Algorithms for the ATSP

3.1 Introduction

The Traveling Salesman Problem (TSP) is the problem of finding a shortest tour through a given number of locations such that every location is visited exactly once. The cost of traveling from location i to location j is denoted by $c(i, j)$. These costs are called *symmetric* if $c(i, j) = c(j, i)$ for each pair of cities i and j, and *asymmetric* otherwise. The fact that the TSP is a typical $\mathcal{NP}$-hard optimization problem means, roughly spoken, that solving instances with a large number of cities is very difficult if not impossible. Recent developments in polyhedral theory and heuristics have significantly increased the size of instances which can be solved to optimality. The best known exact algorithms are based on either the Branch and Bound (BnB) method for the Asymmetric TSP (ATSP) (Fischetti *et al.*, 2002) or the Branch-and-Cut method for the Symmetric TSP (STSP) using the double index formulation of the problem (see Naddef (2002)).

Currently, most algorithms for the TSP delete high cost arcs or edges and save the low cost ones. A drawback of this strategy is that costs of arcs and

[0]This chapter is based on the article: M. Turkensteen, D. Ghosh, B. Goldengorin, and G. Sierksma,"Tolerance-Based Branch and Bound Algorithms for the ATSP", accepted for publication in EJOR. A preliminary version of this paper has been published in B. Goldengorin, G. Sierksma, M. Turkensteen, "Tolerance Based Algorithms for the ATSP", Graph-Theoretic Concepts in Computer Science. 30th International Workshop, WG 2004, Bad Honnef, Germany, June 21-23, 2004. Hromkovic J., Nagl M., Westfechtel B. (eds.), Lecture Notes in Computer Science 3353, pp. 222–234, (2004).

edges are no accurate indicators whether those arcs or edges are saved in an optimal TSP solution. In this paper, it is shown that *tolerances* are better indicators. A tolerance value of an edge/arc is the cost of excluding or including that edge/arc from the solution at hand; see Section 3.3. Although the concept of tolerances has been applied for decades (in sensitivity analysis; see for example Libura *et al.* (1998); Lin and Wen (2003)), only Helsgaun's version of the Lin-Kernighan heuristic for the STSP applies tolerances; see Helsgaun (2000).

We apply upper tolerances in BnB algorithms for the ATSP. A BnB algorithm initially solves a *relaxation* of the original hard problem. In case of the ATSP, the Assignment Problem (AP) is a common choice. The AP is the problem of assigning n people to n jobs against minimum cost; an optimal solution of the AP is called a *minimum cycle cover*. If the minimum cycle cover at hand is a complete tour, then the ATSP instance is solved; otherwise, the problem is partitioned into new subproblems by including and excluding arcs. In the course of the process, a *search tree* is generated in which all solved subproblems are listed. BnB algorithms comprise two major steps: *branching* and *bounding*.

The objective of *branching* is to find a good, or even optimal, ATSP solution in an effective way. If the current AP solution is infeasible, then there may exist a subset of elements of this solution, the so-called *survival set*, which also appears in an optimal TSP solution eventually obtained by the BnB algorithm. An effective BnB algorithm cherishes arcs in survival sets and disposes of the other ones, the *extinction arcs*. Obviously, survival sets are not known beforehand. Predictions of what arcs belong to the survival set are usually based on the arc costs. We claim that the predictions are much more accurate if upper tolerance values of arcs are used instead; see Section 3.5.

The objective of *bounding* is to fathom as many nodes in the search tree as possible. A subproblem is fathomed if its lower bound exceeds the value of the best solution found so far in the process. An AP solution is infeasible for the ATSP, if it consists of two or more subcycles; we call such subcycles *offenders*. To obtain a complete tour, at least one offender must be "broken", meaning that its arcs are successively prohibited in the next stage of the process. Since the cost of removing an arc is its upper tolerance value, upper tolerance values provide us with the cost of breaking an offender, and hence, they can be used to tighten the lower bounds. The higher the lower bound, the larger the set of subproblems

that are fathomed; see Section 3.6.

Compared to their cost-based counterparts, tolerance-based BnB algorithms have one big drawback: whereas cost values need to be looked up, tolerance values must be calculated. So the question is whether the reduction in the size of the search tree is on average sufficiently large to compensate for the additional tolerance computation times. Computational experiments, performed in Section 3.9, show that it is so for random, sparse, and various ATSPLIB instances. The conclusions and future research directions appear in Section 3.10.

3.2 Branch and Bound Algorithms for the Asymmetric Traveling Salesman Problem

ATSP instances are often solved to optimality with BnB algorithms that take the Assignment Problem (AP) as a relaxation (Fischetti *et al.*, 2002). Instead of a single tour through all cities, an optimal solution of the AP usually consists of more than one tour, the so-called *subcycles*. The AP-based BnB algorithms then remove the arcs in a chosen subcycle one by one, combining the subcycles into an optimal tour H^* of the given instance. BnB algorithms are built up from the following four basic ingredients; see for example Miller and Pekny (1991).

The *branching rule* prescribes how the current problem should be partitioned into subproblems. An effective branching rule for the ATSP with the AP relaxation is introduced in Carpaneto and Toth (1980).

The *search strategy* prescribes which subproblems should be expanded next. Two widely used strategies are *Depth First Search (DFS)* which solves the most recently generated subproblem first, and *Best First Search (BFS)* which solves the most promising subproblem first, i.e., the subproblem with the lowest value of the AP bound.

The *upper bounding strategy* prescribes how tours should be constructed in the BnB process. A method to construct feasible ATSP tours from AP solutions is the patching procedure by Karp and Steele (1990).

The *lower bounding strategy* determines how a lower bound of the solution value of any subproblem should be constructed. The value of the AP solu-

tion of the subproblem is usually taken as a lower bound.

State-of-the-art BnB algorithms for the ATSP can be found in Miller and Pekny (1991) and Carpaneto *et al.* (1995). These algorithms apply patching to obtain upper bounds, use AP lower bounds, and branch on a smallest cycle in the current AP solution, i.e., a cycle of smallest cardinality. The search strategy of both algorithms is BFS. This means that for many ATSP instances, solutions are obtained in very short solution times. On the other hand, a list of subproblems should be maintained in order to determine the most promising one. As a consequence, BFS BnB algorithms tend to run out of memory when the search trees grow large. For example, Miller and Pekny (1991) report that their BnB algorithm cannot solve symmetric instances of size as small as 30! The DFS strategy has two advantages over BFS algorithms: less memory overhead is required to the store data on the search tree, and the relaxation solution of the parent node in the search can be used to obtain a relaxation solution at the current node more quickly. As a consequence, it can be expected that DFS algorithms are better applicable to difficult instances. For this reason, we consider DFS algorithms in this paper, and compare the performance of our DFS algorithms to the algorithm by Carpaneto *et al.* (1995).

3.3 Tolerances for Combinatorial Optimization Problems

In this section, we introduce the notion of upper and lower tolerances in case of a general Combinatorial Optimization Problem (COP). A *Combinatorial Optimization Problem* COP$(\mathcal{E}, C, \mathcal{D}, f_C)$ is the problem of finding a solution

$$S^* \in \arg\, opt\{f_C(S) \mid S \in \mathcal{D}\},$$

where $C : \mathcal{E} \rightarrow \Re$ is the given *instance* of the problem with *ground set* $\mathcal{E}$ satisfying $|\mathcal{E}| = m$ $(m \geq 1)$, $\mathcal{D} \subseteq 2^{\mathcal{E}}$ is the *set of feasible solutions*, and $f_C : 2^{\mathcal{E}} \rightarrow \Re$ is the *objective function* of the problem. $\mathcal{D}^* = \arg\, opt\{f_C(S) \mid S \in \mathcal{D}\}$ is the set of optimal solutions. It is assumed that $\mathcal{D}^* \neq \emptyset$. In the remaining part of this paper we take $opt = \min$.

Let $g \in \mathcal{E}$, and $\alpha \geq 0$. By $C_{\alpha,g} : \mathcal{E} \rightarrow \Re$ we denote the instance defined as $C_{\alpha,g}(e) = C(e)$ for each $e \in \mathcal{E} \setminus \{g\}$, and $C_{\alpha,g}(g) = C(g) + \alpha$. Take any

$S^* \in \mathcal{D}^*$, and $e \in \mathcal{E}$. The *upper tolerance* of e with respect to S^* is denoted and defined as

$$u_{S^*}(e) = \max\{\alpha \geq 0 \,:\, S^* \in \arg\min\{f_{C_{\alpha,e}}(S) \,:\, S \in \mathcal{D}\}\},$$

and the *lower tolerance* of e with respect to S^* as

$$l_{S^*}(e) = \max\{\alpha \geq 0 \,:\, S^* \in \arg\min\{f_{C_{-\alpha,e}}(S) \,:\, S \in \mathcal{D}\}\}.$$

I.e. $u_{S^*}(e)$ is the maximal increase of $C(e)$ under which S^* stays optimal, and $l_{S^*}(e)$ the maximal decrease of $C(e)$ under which S^* stays optimal.

We assume that f_C is *monotone*, meaning that for each $S \in 2^{\mathcal{E}}$ and each $\alpha \geq 0$, it holds that

$$f_{C_{\alpha,e}}(S) \geq f_{C_{0,e}}(S).$$

Sum functions with $f_C(S) = \sum_{e \in S} C(e)$, *bottleneck functions* with $f_C(S) = \max_{e \in S} C(e)$, and *product functions* with $f_C(S) = \prod_{e \in S} C(e)$ and $C(e) \geq 1$ for each $e \in \mathcal{E}$ are all monotone functions.

We call the set $\mathcal{D}$ of feasible solutions *non-embedded* if for each $S_1, S_2 \in \mathcal{D}$ with $S_1 \neq S_2$, it holds that neither $S_1 \subset S_2$ nor $S_2 \subset S_1$.

The following theorem, of which the proof is left to the reader, can be seen as a straightforward generalization of Libura's theorem on tolerances (see Libura (1991)) for the TSP. We will use the following extra notations. Let $e \in \mathcal{E}$. Then $\mathcal{D}_+(e) = \{S \in \mathcal{D} \,:\, e \in S\}$, and $\mathcal{D}_-(e) = \{S \in \mathcal{D} \,:\, e \notin S\}$. Clearly, $\mathcal{D} = \mathcal{D}_-(e) \cup \mathcal{D}_+(e)$ and $\mathcal{D}_-(e) \cap \mathcal{D}_+(e) = \emptyset$. $\mathcal{D}^*_+(e)$ and $\mathcal{D}^*_-(e)$ are the sets of optimal solutions containing e and not containing e, respectively.

Theorem 3.3.1. *Consider COP $(\mathcal{E}, C, \mathcal{D}, f_C)$ with monotone f_C. For each $S^* \in \mathcal{D}^*$, the following holds:*

1. $e \in \cap\mathcal{D}^*$ *iff*
   $u_{S^*}(e) = f_C(S) - f_C(S^*) > 0$
   for each $S \in \mathcal{D}^*_-(e)$,
   $l_{S^*}(e) = \infty$;
2. $e \in \mathcal{E} \setminus \cup\mathcal{D}^*$ *iff*
   $u_{S^*}(e) = \infty$, $l_{S^*}(e) = f_C(S) - f_C(S^*) > 0$
   for each $S \in \mathcal{D}^*_+(e)$;
3. $e \in S^* \setminus \cap\mathcal{D}^*$ *iff*
   $u_{S^*}(e) = 0$, $l_{S^*}(e) = \infty$;
4. $e \in \cup\mathcal{D}^* \setminus S^*$ *iff*
   $u_{S^*}(e) = \infty$, $l_{S^*}(e) = 0$.

If $|\mathcal{D}^*| = 1$, then this theorem boils down to Libura's theorem on tolerances. If $\mathcal{D}_-(e) = \emptyset$ for some $e \in \mathcal{E}$, then $u_{S^*}(e) = \min\{f_C(T) : T \in \mathcal{D}_-(e)\} - f_C(S^*) = \min\{\emptyset\} = \infty$ (by definition). Similarly, for $\mathcal{D}_+(e) = \emptyset$ we take $l_{S^*}(e) = \infty$.

The following statement can be derived from Theorem 3.3.1. If one excludes an element e from S^*, then the objective value of the new problem will be $f_C(S^*)+u_{S^*}(e)$. The same holds for the lower tolerance if the element $e \in \mathcal{E}\setminus S^*$ is included. So a tolerance-based BnB algorithm knows the cost of including or excluding elements before it selects the element to branch on.

3.4 Upper Tolerances of the Assignment Problem

In this section, we introduce the tolerance values of the Assignment Problem (AP). Recall that the AP is the problem of assigning n employees to n jobs against minimum costs, given a cost matrix C. Each employee is only allowed to perform one job. The assignment of employee i to job j is denoted by the arc (i, j) and has cost $c(i, j)$. An *instance* of the AP is defined by its cost matrix C. Let $\mathcal{A}$ denote the set of feasible solutions of the AP instance C, and let $\mathcal{A}^*$ be the set of optimal solutions of C. We denote the cost of the assignment $A \in \mathcal{A}$ of instance C by $f_C(A)$. The cost is obtained by adding the costs of all arcs $e \in A$, so $f_C(A) = \sum_{e\in A} c(e)$. The solution of the AP can be represented as a set of cycles, and the AP solution is also called a *minimum cycle cover*.

Consider the AP instance with the following cost matrix C, borrowed from Balas and Toth (1985).

City	1	2	3	4	5	6	7	8
1	∞	2	11	10	8	7	6	5
2	6	∞	1	8	8	4	6	7
3	5	12	∞	11	8	12	3	11
4	11	9	10	∞	1	9	8	10
5	11	11	9	4	∞	2	10	9
6	12	8	5	2	11	∞	11	9
7	10	11	12	10	9	12	∞	3
8	10	10	10	10	6	3	1	∞

The (unique) optimal AP solution A^* consists of the three cycles $K_1 = \{(1,2),(2,3),(3,1)\}$, $K_2 = \{(4,5),(5,6),(6,4)\}$, and $K_3 = \{(7,8),(8,7)\}$. The cost value of this solution is 17.

Take any $A^* \in \mathcal{A}^*$, while A^* need not be unique. The upper tolerance value of any arc e is the maximum increase in the cost $c(e)$ such that A^* remains optimal. More formally, let $f_C(A^*)$ denote the cost of any assignment solution A^* of the instance C, and let $C_{\alpha,e}$ denote the instance in which the cost value of arc e is increased with α in comparison with the instance C, and the costs of all other arcs remain unchanged. The upper tolerance value of arc e with respect to A^* is denoted by and defined as:

$$u_{A^*}(e) = \max\{\alpha \geq 0 \,:\, A^* \in \arg\min\{f_{C_{\alpha,e}}(A) \,:\, A \in \mathcal{A}^*\}\},$$

Upper tolerances are defined with respect to a fixed optimal solution A^*, because A^* need not be unique. Consider an AP instance with two optimal solutions A_1^* and A_2^*, and let arc $e \in A_1^* \setminus A_2^*$. Then, by definition, $u_{A_1^*}(e) = 0$, whereas $u_{A_2^*}(e) = \infty$.

Section 3.3 show that the upper tolerance value of the arc e corresponds to the value of the cheapest solution without e; the proof is based on Libura (1991). More formally, let $C_{\infty,e}$ be the instance from which e is excluded, let $\mathcal{A}^*_-(e)$ be the set of optimal solutions of $C_{\infty,e}$, and let $A^*_-(e) \in \mathcal{A}^*_-(e)$. The instance $C_{\infty,e}$ is formed by setting the cost value of e to a very large number. The cost of the optimal solution $A^*_-(e)$ of the new instance is $f_C(A^*_-(e))$. We may write C instead of $C_{\infty,e}$, since $e \notin A^*_-(e)$. The upper tolerance of e satisfies $u_{A^*}(e) = f_C[A^*_-(e)] - f_C[A^*]$. In order to compute one upper tolerance value, the additional AP instance $C_{\infty,e}$ needs to be solved.

Consider the AP example above. The upper tolerance of the arc $(7,8)$ is obtained by setting the entry $c(7,8)$ to ∞ and solving the newly obtained in-

stance. The optimal solution of the new instance contains the arcs $(1, 8)$ and $(7, 1)$ instead of $(7, 8)$ and $(1, 2)$. The cost of this solution $f_C[A^*_-(7, 8)] = 28$. So the upper tolerance value satisfies: $u_{A^*}[(7, 8)] = f_C[A^*_-(7, 8)] - f_C(A^*) = 28 - 17 = 11$.

Although solving an AP from scratch takes $O(n^3)$ time, it is well known (see, e.g., Balas and Toth (1985)) that for finding an optimal solution $A^*_-(e)$ based on the given AP solution A^*, only one labelling procedure in the Hungarian method needs to be performed, which can be done in $O(n^2)$ time.

3.5 Survival Sets

This section explores the *branching* step of BnB algorithms. The goal of branching is to find a good or even optimal solution in the fastest possible way. BnB methods generate sequences of steps in which parts of the solution at hand are included and excluded, until an optimal solution of the original problem is found. If a BnB algorithm predicts correctly which element to delete or to insert, then its search tree will be small. So it is important to predict survival sets accurately. Most algorithms base the predictions on cost values, but the question is: do predictions improve if they are based on upper tolerance values, and: how many survival arcs are there on average?

We have selected instances from the ATSPLIB (see Reinelt (1991)) of size $n = 34$ to 171. In our experiments we consider instances with varying degree of symmetry and degree of sparsity. The *degree of symmetry* is defined as the fraction of off-diagonal entries of the cost matrix $\{c_{ij}\}$ that satisfy $c_{ij} = c_{ji}$; the *degree of sparsity* is the percentage of arcs that is missing in an instance. The random instances have degree of symmetry 0, 0.33, 0.66, and 1. The sparse random instances have degree of sparsity of 50%, 75%, and 90%. All randomly generated instances have size 60, 70, and 80. Finally, the instances by Buriol *et al.* (2004) are derived from symmetric TSPLIB instances. The almost symmetric Buriol instances are derived from symmetric instances from the TSPLIB as follows; see Buriol *et al.* (2004). Let σ be the average of all distances of the STSPLIB instance, and let k' be a user-defined number. Each intercity distance on the lower diagonal of the cost matrix C of the original symmetric instance is increased by a factor $k\sigma$, where k is randomly drawn from the uniform distribu-

Table 3.1. Fraction of survival arcs in optimal AP and ATSP solutions

Instance type	Fraction of survival arcs
ATSPLIB	53.52%
Degree of symmetry 0.33	69.29%
Degree of symmetry 0.66	51.10%
Full symmetry	43.44%
Asymmetric random	80.49%
Degree of sparsity 50%	86.27%
Degree of sparsity 75%	84.23%
Degree of sparsity 90%	83.46%
Buriol, $k' = 5$	46.72%
Buriol, $k' = 50$	72.95%

tion supported on $\{0, \ldots, k'\}$. So the value of k' is a measure of the deviation of the instances from full symmetry.

Table 3.1 shows that the average percentage of common arcs in corresponding AP and ATSP solutions varies between 40 and 80%. Similar investigations show that the Minimum 1-Trees and optimal STSP tours have between 70% and 80% of the edges in common (Helsgaun, 2000).

Let $\mathcal{H}$ denote the set of all feasible tours of an ATSP instance, and define $\mathcal{H}^*$ as the set of optimal tours. Note that $\mathcal{H} \subseteq \mathcal{A}$. Assume that we start with a fixed AP solution $A^* \in \mathcal{A}^*$, and that $H^* \in \mathcal{H}^*$ is a fixed shortest complete tour of the same instance. We explore whether there are relationships between the cost values and the upper tolerance values of arcs and their appearance in H^*. These relationships are measured with the *adjusted Rand index*, which measures the relationship between two partitions; see Hubert and Arabie (1985), and with *logistic regression* (Gessner *et al.*, 1988). The costs and the upper tolerances are continuous variables, and are both compared with the partition of A^* into survival and extinction arcs.

In the adjusted Rand index analysis (Hubert and Arabie, 1985), we create partitions based on upper tolerances and costs. First, the arcs in a fixed optimal AP solution A^* are partitioned into two subsets: IN_1 contains the survival arcs, and the subset IN_0 contains the extinction arcs. Call this partition $IN = \{IN_0, IN_1\}$. We try to replicate IN with partitions C and U based on the costs and tolerance values of the arcs, respectively. Define $C := \{C_0, C_1\}$,

Table 3.2. Quality of the predictions using upper tolerances and costs

	Adjusted Rand indices		R^2 of logit model	
Instance type	Tolerance	Cost	Tolerance	Cost
ATSPLIB	0.113	-0.003	0.112	0.035
Degree of symmetry 0.33	0.152	0.007	0.169	0.017
Degree of symmetry 0.66	0.188	0.028	0.132	0.015
Full symmetry	0.158	0.013	0.007	0.011
Asymmetric random	0.287	0.039	0.299	0.023
Sparsity 50%	0.361	0.017	0.407	0.015
Sparsity 75%	0.252	0.033	0.382	0.013
Sparsity 90%	0.219	0.032	0.342	0.017
Buriol, $k' = 5$	-0.019	-0.023	0.010	0.006
Buriol, $k' = 50$	0.217	0.0303	0.172	0.028

where $C_0 = \{e \in A^* : c(e) \geq c^*\}$, $C_1 := \{e \in A^* : c(e) < c^*\}$, and determine c^* in such a way that $|C_0| = |IN_0|$. Arcs are partitioned into a set of low cost arcs C_1 and a class of high cost arcs C_0. If it is true that all high cost arcs are not in the given shortest tour, then the sets IN_0 and C_0 and the sets IN_1 and C_1 coincide and cost values lead to a perfect prediction. Similarly, define $U = \{U_0, U_1\}$, where $U_0 := \{e \in A^* : u_{A^*}(e) < u^*\}$, $U_1 := \{e \in A^* : u_{A^*}(e) \geq u^*\}$, and determine u^* in such a way that $|U_0| = |IN_0|$.

The *adjusted Rand index* measures how similar the each of both partitions U and C are in comparison with the ideal partition into survival and extinction arcs IN. The formula of the adjusted Rand index is given in Section 5.4. The more similar two partitions are, the higher the adjusted Rand index between both partitions is. An adjusted Rand index of 1 indicates that for each nonempty class A_i of partition A, there exists a class B_j of partition B such that $A_i = B_j$. The expected adjusted Rand index is 0, if both partitions assign objects to classes randomly having the original number of objects in each class (Hubert and Arabie, 1985).

The adjusted Rand indices between IN and C and between IN and U are shown in Table 3.2. They are larger for the tolerance-based partitions U than for the cost-based partitions C, which indicates that predictions are better if they are based on upper tolerance values.

The adjusted Rand index analysis makes splits in the data based on the cost

and the upper tolerance values. It does not include the distances in cost or tolerance value of an arc from the split value. The following problem arises. Suppose that an arc with a very high upper tolerance value is not in any shortest ATSP tour. Such an arc is very likely to be excluded already in an early stage of a tolerance-based branching process. If this event occurs, the predictions of upper tolerances should get a bad rating for this instance. The same holds for arcs with low cost values which are not in a shortest ATSP tour. We define the binary variable IN, ranging over all arcs $e \in A^*$, as follows: $IN(e) = 1$ if $e \in H^*$ and $IN(e) = 0$ otherwise.

An appropriate method for estimating the relationship between a dependent binary variable and independent continuous variables is *logistic regression* (Hair *et al.*, 1998). Logistic regression is usually applied to explain or predict choices in choice modeling, for example the choice between buying and not buying a product (Hosmer and Lemeshow, 1989). Based on the values of the independent variables, probabilities $\pi(e)$ are estimated of the event that the independent variable attains the value 1 for the observation e. The fit of a model is good if these probabilities $\pi(e)$ are close to the actual observed values of the dependent variable.

A general measure to determine the fit of a logistic regression model, also called *logit model*, is the R^2 *for a logit model* R^2_{logit} (Gessner *et al.*, 1988), which compares the predictive power of a logit model to the predictive power of a model without independent variables. If $R^2_{logit} = 1$, then all the variance in the independent variable is explained and predictions are perfect. On the other hand, if $R^2_{logit} = 0$, the independent variables in the model offer no information about the dependent variable. R^2_{logit} is similar to R^2 in linear regression.

In order to analyze survival sets, we construct for each instance two separate logit models. In the cost-based model the dependent variable IN is explained by the cost values of the arcs in an assignment solution; the independent variable in the tolerance-based model is formed by the upper tolerance values. The values of R^2_{logit} of both models are presented in Table 3.2.

The values of R^2_{logit} are higher for the tolerance-based models, except for fully symmetric instances. These results confirm that predictions based on upper tolerance values are clearly better than predictions based on costs.

3.6 Tolerance-Based Lower Bounds for the ATSP

In this section, we use the upper tolerances of an optimal AP solution to construct tight lower bounds for the corresponding ATSP. These lower bounds are introduced in Goldengorin *et al.* (2004). Recall that, if the lower bound of a subproblem is increased, then this subproblem is more likely to be fathomed during the execution of the BnB algorithm.

In BnB algorithms, lower bounds can be obtained by removing sources of infeasibility with respect to the original problem from the current solution. We call such sources of infeasibility *offenders*. In case of the ATSP and its AP relaxation, offenders are subcycles. Let A^* consist of $k(>1)$ cycles, say, $A^* = \cup_{i=1}^{k} K_i$. We define $\mathcal{C}(A^*)$ as the set of all cycles in A^*, so $\mathcal{C}(A^*) = \{K_1, \ldots, K_k\}$.

In order to "break" a subcycle K (meaning that this cycle does not appear in subsequent AP solutions), at least one arc must be removed. Recall that the cost of removing an arc is equal to its upper tolerance value. Hence, the minimum cost of breaking a subcycle is equal to the lowest upper tolerance value in that cycle. We denote and define for each $K \in \mathcal{C}(A^*)$ this value by $u_{A^*}^K := \min\{u_{A^*}(e) : e \in K\}$. Theorem 3.6.1 shows that the cost of breaking a cycle by deleting a minimum tolerance arc can be used to increase the lower bound.

Theorem 3.6.1. *Let A^* and H^* be optimal solutions to AP and ATSP instances, respectively, with the same cost matrix C. Assume that A^* consists of at least two cycles. Then for each $K \in \mathcal{C}(A^*)$, the following inequalities hold:*

$$f_C(A^*) \leq f_C(A^*) + u_{A^*}^K \leq f_C(H^*).$$

Proof. The first inequality is obvious, since $u_{A^*}(e) \geq 0$ for every $e \in A^*$. Now we show that $f_C(A^*) + u_{A^*}^K \leq f_C(H^*)$. Take any $K \in \mathcal{C}(A^*)$, and take any $e \in K \setminus H^*$. Let $\mathcal{A}_-(e)$ be the set of all AP solutions without e, so $\{A \in \mathcal{A} : e \notin A\}$. Moreover, let $A_-^*(e) \in \arg\min\{f_C(A) : A \in \mathcal{A}_-(e)\}$. From Section 3.3, we have that $u_{A^*}(e) = f_C[A_-^*(e)] - f_C(A^*)$. Since $H^* \in \mathcal{A}_-(e)$ $f_C(A^*) + u_{A^*}(e) = f_C[A_-^*(e)] \leq f_C(H^*)$. Hence $f_C(A^*) + u_{A^*}^K \leq f_C(A^*) + u_{A^*}(e) = f_C[A_-^*(e)] \leq f_C(H^*)$. □

Based on Theorem 3.6.1, we may ask the question which subcycle in A^* should be "broken". The most effective choice is the cycle K in which $u_{A^*}^K$ is

maximal, since it causes the largest increase in the lower bound $f_C(A^*) + u^K_{A^*}$. However, all tolerance values in all cycles must be computed to guarantee that we obtain an arc with maximal value of $u^K_{A^*}$. This is usually a time-consuming matter. Hence, it may be worthwhile to restrict the tolerance calculations to a 'not too large' subset of $\mathcal{C}(A^*)$. Let $O \subseteq \mathcal{C}(A^*)$. Denote and define the *bottleneck tolerance* with respect to $O \subseteq \mathcal{C}(A^*)$ by $bu_{A^*}(O) := \max\{u^K_{A^*} : K \in O\}$, and the corresponding *bottleneck bound* by $lb_{A^*}(O) = f_C(A^*) + bu_{A^*}(O)$. The choice for the set of offenders O determines the values of the bottleneck tolerances and bounds. For example, if $\mathcal{C}(A^*) = \{K_1, K_2\}$, $u^{K_1} = 1$, and $u^{K_2} = 5$, then $bu_{A^*}(\mathcal{C}(A^*)) = bu_{A^*}(\{K_1\}) = 5$, whereas $bu_{A^*}(\{K_2\}) = 1$ and $bu_{A^*}(\emptyset) = 0$.

We consider special sets of offenders in $\mathcal{C}(A^*)$:

The *Entire Cycle Set (ECS)*. Define $O_E := \mathcal{C}(A^*)$. So the bottleneck tolerance value in the entire set of subcycles of A^* is taken. Note that $bu_{A^*}(O_E) \geq bu_{A^*}(O)$ for every $O \subseteq \mathcal{C}(A^*)$ and for every A^*. This lower bound corresponds with the Exact Bottleneck Bound from Goldengorin *et al.* (2004).

The *Smallest Cycle Set (SCS)*. Define $O_S := \{K^*\}$, where K^* is a cycle of A^* with the smallest cardinality, i.e., $K^* \in \arg\min\{|K| : K \in \mathcal{C}(A^*)\}$. This lower bound corresponds to the Approximate Bottleneck Bound from Goldengorin *et al.* (2004).

The concept of bottleneck tolerances uses the structure of an assignment solution to increase the lower bound. For instance, suppose that an assignment solution of a randomly generated instance consists of many small cycles. We may expect a high bottleneck tolerance value, since the maximum is taken from a large set of numbers. Note also that if an assignment consists of a large number of cycles, then, on average, the gap between the AP and the ATSP solution values is wide. So there is a relationship between the size of the gap and the value of the bottleneck tolerance.

The ECS choice leads to the tightest upper tolerance-based lower bounds. The SCS choice is taken into account, because it gives a good approximation for the ECS bound in a short time. We claim that, in many situations, the value of $bu_{A^*}(O_E)$ is attained on a smallest cycle, and hence, $bu_{A^*}(O_S)$ is a good approximation of it. The intuition behind this claim is the following. Suppose

upper tolerance values are randomly dispersed over the arcs of a minimum cycle cover. The minimum tolerance value of a small cardinality cycle is then relatively large; therefore, it is likely that $bu_{A^*}(O_E)$ is attained on a smallest cycle of A^*. Table 3.3 shows that the fraction of subproblems in a BnB search tree for which this event occurs, is about 45%.

Table 3.3. Percentage $bu_{A^*}(O_E)$ in smallest cycles and reductions by ECS and SCS lower bounds

Instance	$bu_{A^*}(O_E)$ in smallest cycles	$r(O_E)$	$r(O_S)$
ATSPLIB	46.38%	19.97%	6.39%
Degree of symmetry 0.33	60.98%	34.62%	17.07%
Degree of symmetry 0.66	69.48%	26.66%	12.27%
Full symmetry	88.49%	21.64%	14.61%
Asymmetric random	43.09%	50.47%	43.31%
Degree of sparsity 50%	43.73%	56.50%	48.99%
Degree of sparsity 75%	44.06%	45.78%	40.45%
Degree of sparsity 90%	44.49%	49.86%	35.45%

The next natural question is: what is the difference in quality between $lb_{A^*}(O_C)$ and $lb_{A^*}(O_S)$? To measure the quality of a lower bound, we introduce the *reductions* $r(O)$ of the gap between $f_C(A^*)$ and $f_C(H^*)$ achieved by the lower bound $lb_{A^*}(O)$. Define $r(O) = \frac{bu_{A^*}(O)}{f_C(H^*)-f_C(A^*)} \times 100\%$. Table 3.3 compares $r(O_E)$ and $r(O_S)$. The results show that, for (quasi-)symmetric and ATSPLIB instances, the ECS choice clearly constructs better lower bounds than the SCS choice. However, the SCS choice gives a satisfactory approximation for asymmetric random and sparse instances, while it is generally much faster to compute.

In BnB settings, lower bounds can be computed at every node of the search tree. A high quality bound, such as $lb_{A^*}(O_E)$ allows the BnB algorithm to discard a large number of nodes, but it requires long computing times at each node considered. In order to find the balance between bound quality and computing times, computational experiments are performed in Section 3.9.

The approaches are clarified with the ATSP example from Section 3.4. Recall that the unique AP solution A^* consists of the three cycles $K_1 = \{(1,2),(2,3),$

$(3,1)\}$, $K_2 = \{(4,5),(5,6),(6,4)\}$, and $K_3 = \{(7,8),(8,7)\}$. Moreover, $f_C(A^*) = 17$ and $f_C(H^*) = 26$. Figure 3.1 depicts A^* and the upper tolerance values of the arcs.

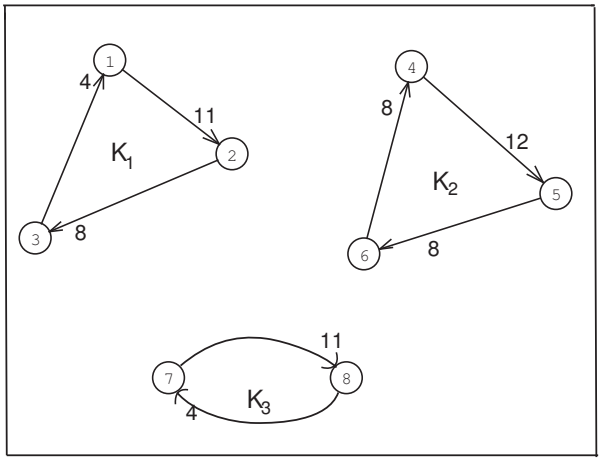

Figure 3.1. Minimum cycle cover with arc upper tolerances

In this example, $u^{K_1} = 4$, $u^{K_2} = 8$, and $u^{K_3} = 4$. So $bu_{A^*}(O_E) = 8$ is attained on the arcs $(5,6)$ and $(5,6)$ in cycle K_2, and $bu_{A^*}(O_S) = 4$ on arc $(8,7)$ in the smallest cycle K_3. Therefore, $lb_{A^*}(O_S) = 21$ and $lb_{A^*}(O_E) = 25$. Since $f_C(H^*) = 26$, $r(O_E) = \frac{8}{26-17} \times 100\% = 88.8\%$, $r(O_S) = \frac{4}{26-17} \times 100\% = 44.4\%$. For this instance, $lb_{A^*}(O_E) = 25$ is tighter than all other bounds discussed in Balas and Toth (1985).

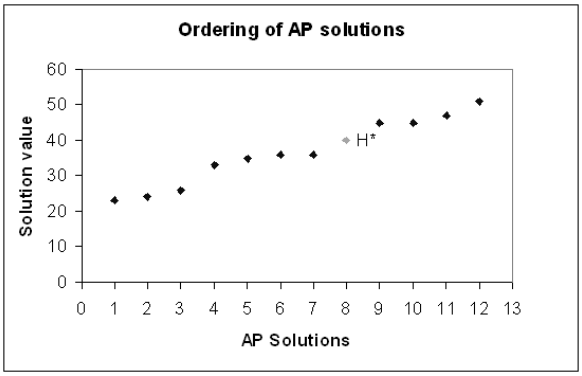

Figure 3.2. Enumeration of AP solutions

Applying the concept of bottleneck tolerances not only increases lower

bounds, but it also strengthens the branching. The previous section shows that it is worthwhile to branch on an arc with a small upper tolerance value. But which one should we choose? Figure 3.2 depicts an enumeration of feasible solutions of an AP instance in a non-decreasing order of cost values. A careful branching strategy, which branches on a smallest upper tolerance arc, obtains all AP solutions with cost smaller than $f_C(H^*)$. However, if an algorithm branches on a bottleneck tolerance arc, then it traverses the enumeration with larger steps, and it is likely to arrive at H^* in fewer branching steps. That is, of course, if it does not exclude survival arcs. Table 3.3 indicates that the exclusion of an ECS bottleneck arc from an asymmetric, randomly generated instance brings the solution value of the next subproblem on average about 50% closer to $f_C(H^*)$. We propose branching rules based on bottleneck tolerances arcs obtained by the ECS and the SCS choices. We call these the *ECS* and *SCS branching rules*, respectively.

3.7 The algorithms

In the previous sections, have discussed tolerance-based branching rules and lower bounds for the ATSP. In Goldengorin *et al.* (2004), it is shown that search tree reductions of tolerance-based DFS algorithms are the largest when the branching rule and the lower bound act in conjunction. We explain this in the next section with the so-called *synergy effect*. Two tolerance-based algorithms are considered in the experiments: $BnB(ECS)$ uses the ECS branching rule and lower bound, whereas $BnB(SCS)$ uses the SCS branching rule and lower bound. whereas $BnB(SCS)$ uses the SCS branching rule and lower bound. After computing an assignment solution, the upper tolerance values of arcs in the specified cycle set are determined for the lower bound. When the subproblem cannot be discarded, these upper tolerance values are used for branching. The cost-based benchmark $BnB(Cost)$ is a DFS algorithm which branches on the longest arc in a smallest cycle of the current AP solution. The algorithms apply the subtour elimination scheme from Carpaneto and Toth (1980), and the reduction procedure from Carpaneto *et al.* (1995) is used at the top node of the search tree to determine which arcs will never appear in an optimal solution. These arcs are then removed from the arc set. Moreover, the algorithms apply the solver from

Jonker and Volgenant (1986) to solve the APs and to compute the upper tolerance values. These algorithms are proved to be competitive in the AP solver comparison by Dell'Amico and Toth (2000).

The flowchart in Figure 3.3 provides a schematic view of a tolerance-based BnB algorithm. In each subproblem of $BnB(ECS)$, n upper tolerances are computed where n is the dimension of the current subproblem. $BnB(SCS)$ uses the SCS branching rule and lower bound. Hence, the complexities are $O(n^3)$ and $O(|K^*|n^2)$, respectively. The ECS branching rule selects a subcycle in which the largest minimum upper tolerance value is achieved. Subsequently, it branches on the arcs in this cycle in non-decreasing order of tolerances.

The performance of our DFS methods is compared to the performance of the CDT algorithm by Carpaneto *et al.* (1995). In Fischetti *et al.* (2002), the CDT code is not able to solve the ATSPLIB instances ft53, ftv170, kro124p, p43, and ry48p within the specified time limit of 1,000 seconds. Fischetti *et al.* (2002) also note that the CDT approach "either solve the problems within CPU 1,000 seconds, or the final gap is too large to hope in a convergence within 10,000 seconds". By setting the maximum computing time to 10,000 seconds, the search trees are kept reasonably small. Our Pentium 4 computer is approximately twice as fast as the Alpha 500 MHz used for the experiments in Fischetti *et al.* (2002); see the machine speed comparison from Johnson (2006). The BnB algorithms are listed in Table 3.4.

Table 3.4. BnB algorithms

Algorithm	Description
CDT	BFS algorithm by Carpaneto *et al.* (1995)
$BnB(Cost)$	Cost-based DFS algorithm
$BnB(ECS)$	Algorithm with ECS lower bound and branching rule
$BnB(SCS)$	Algorithm with SCS lower bound and branching rule

3.8 BnB with Tolerance-Based Branching Rules and Lower Bounds

The BnB search trees with tolerance-based lower bounds, with tolerances based branching rules, and with a combination of both are compared in Goldengorin

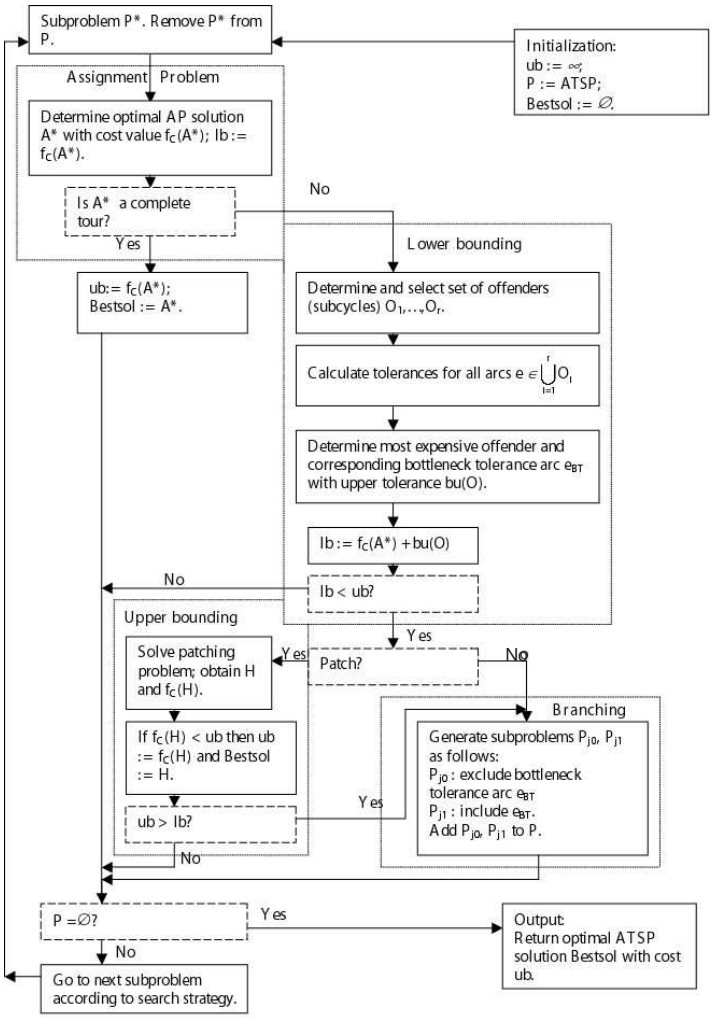

Figure 3.3. Flowchart of a tolerance-based BnB algorithm

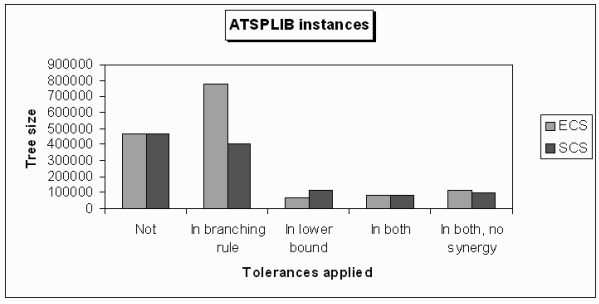

Figure 3.4. Synergy effects for ATSPLIB instances

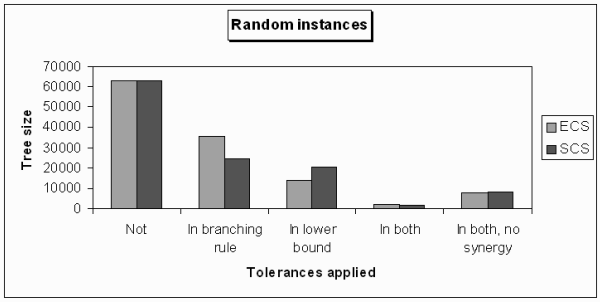

Figure 3.5. Synergy effects for random instances

et al. (2004). They show that BnB algorithms with only new lower bounds have smaller search trees than the conventional algorithm, $BnB(Cost)$. The SCS branching rules also achieves reductions for most instances, but the ECS branching rule only reduces the search trees for random instances. The reductions of the joint use of tolerance-based lower bounds and branching rules are often larger than the reductions when they are used separately.

For many instances, branching on an SCS arc turns out to be more effective than branching on an ECS arc. This is counterintuitive, but the discussion in Section 3.5 may explain this phenomenon. It was observed there that small upper tolerance arcs are more likely to be in an optimal ATSP solution than large upper tolerance arcs. Since the upper tolerance value of an ECS arc is generally higher than the upper tolerance value of an SCS arc, the ECS branching rule is more likely to delete survival arcs than the SCS branching rule. It may also explain why the ECS branching rule leads to larger search trees than the cost-based branching rule in Figure 3.4.

Table 3.5. Search tree sizes with tolerance-based branching rules (BR) and lower bounds (LB) for ATSPLIB instances

			Entire Cycle Set				Smallest Cycle Set			
Instance	n	$BnB(Cost)$	BR	LB	$BnB(ECS)$	$\frac{BnB(Cost)}{BnB(ECS)}$	BR	LB	$BnB(SCS)$	$\frac{BnB(Cost)}{BnB(SCS)}$
ft53	53	20111	$*$	7039	336586		89511	17703	20545	0.98
ft70	70	25831	70047	5619	5861	4.41	22843	6717	4993	5.17
ftv33	34	7065	3867	1983	748	9.45	6007	3137	1569	4.50
ftv35	36	6945	18871	2553	3109	2.23	12047	3219	2265	3.07
ftv38	39	6195	9726	1381	1523	4.07	14663	2821	2387	2.60
ftv44	45	619	976	187	165	3.75	937	249	195	3.17
ftv47	48	29025	29121	8017	3302	8.79	48345	9703	8393	3.46
ftv55	56	92447	98433	12413	10100	9.15	114641	26483	26023	3.55
ftv64	65	43441	89752	9449	8319	5.22	162639	11007	17173	2.53
ftv70	71	253873	459532	25939	50024	5.07	136296	52289	18937	13.41

$*$Memory exhausted

Table 3.6. Search tree sizes with tolerance-based branching rules (BR) and lower bounds (LB) for random instances

		Entire Cycle Set				Smallest Cycle Set			
n	$BnB(Cost)$	BR	LB	$BnB(ECS)$	$\frac{BnB(Cost)}{BnB(ECS)}$	BR	LB	$BnB(SCS)$	$\frac{BnB(Cost)}{BnB(SCS)}$
60	3808	3275	978	323	11.79	1032	2832	221	17.23
70	4528	4085	1138	312	14.51	1286	1781	247	18.33
80	9014	3937	2414	217	41.53	2494	2746	256	35.21
100	9002	3645	1978	174	51.74	2188	5306	135	66.68
200	36390	20497	7114	858	42.41	17612	7612	796	45.72

The search trees of ATSPLIB and random instances are depicted in Figure 3.4. The first column shows the average search tree of $BnB(Cost)$, the second column the trees with tolerance-based branching rules, the third the trees with tolerance-based lower bounds, and the fourth the values of $BnB(ECS)$ and $BnB(SCS)$. If the reductions of tolerance-based lower bounds and branching rules would have been independent, then the actual sizes of the search tree of $BnB(ECS)$ and $BnB(SCS)$ would be equal to their expected sizes, formed by the reduction of the branching rule times the reduction of the lower bound. These values are represented in the column "Both without synergy" in Figures 3.4 and 3.5. However, the actual trees of $BnB(ECS)$ and $BnB(SCS)$ are lower than the expected search trees, indicating that the joint use of tolerance-based lower bounds and branching rules leads to additional search tree reductions. We call this remarkable phenomenon the *synergy effect*. The algorithms $BnB(ECS)$ and $BnB(SCS)$ benefit from the synergy effect.

3.9 Computational Experiments with ATSP Instances

In this section, computational experiments are conducted on the algorithms listed in Table 3.4. The central questions are: do tolerance-based algorithms have smaller search trees, and if this is true, are the reductions sufficient to compensate for the time invested in tolerance computations?

The question we ask now is whether the search tree reductions of $BnB(ECS)$ and $BnB(SCS)$ are sufficient to compensate for the time invested in the tolerance calculations. The cost-based DFS benchmark is $BnB(Cost)$. In order to compare the quality of our DFS methods to BFS methods, we report the solution times of the CDT algorithm on the same computer as well.

We have selected the set of practical ATSPLIB instances by Reinelt (1991). Moreover, we have selected the so-called Buriol instances; the description is given Section 3.5. We solve these practical problems in increasing order of size until the instances encountered cannot be solved within our time limit of 3600 seconds. The random instances have degree of symmetry 0, 0.33, 0.66, and 1, with instance size 60, 70, and 80. The sparse random instances have degree of sparsity of 50%, 75%, and 90%, and their sizes are 100, 200, and 400. For each randomly generated problem set and for all instance sizes, 10 instances

have been generated. The experiments are conducted on a Pentium 4 computer with 256 MB RAM memory and 2 GHz speed. For the ATSPLIB and the Buriol instances, we include three statistics to summarize the results: the number of unsolved instances, the average solution time and the median solution time. The median solution time is included, because the solution times of relatively hard instances are overrepresented and easy instances are underrepresented in the average solution times. The time limit for all algorithms, also CDT, is 3600 seconds; when no optimal solution is found, we report 'Not' in the corresponding table and the instance is considered unsolved. The average and median solution times and search tree sizes contain only instances that are solved by all tested BnB algorithms.

Table 3.7. Search tree sizes and solution times of small ATSPLIB instances

		CDT	$BnB(Cost)$		$BnB(ECS)$		$BnB(SCS)$	
Instance	n	Time	Tree	Time	Tree	Time	Tree	Time
br17	17	3	3674829	24.67	86585	4.18	1034255	22.64
ftv33	34	0	7065	0.22	2195	0.82	1362	0.11
ftv35	36	0	6945	0.22	2307	0.99	1965	0.27
ftv38	39	0	6195	0.22	1381	0.82	2091	0.44
p43	43	Not	Not	3600.00	Not	3600.00	Not	3600.00
ftv44	45	0	619	0.05	453	0.44	171	0.05
ftv47	48	0	29025	1.37	7752	7.20	7692	1.48
ry48p	48	Not	9178227	459.84	183511	262.03	601713	105.88
ft53	53	Not	20111	2.53	336586	448.13	19200	6.15
ftv55	56	1	92447	4.56	17490	24.12	23034	9.12
ftv64	65	1	43441	3.24	9491	20.88	15509	10.77
ft70	70	1	25831	4.01	4123	11.70	4756	1.87
ftv70	71	2	253873	24.07	34838	112.36	17694	8.57
ftv90	91	1	7059	1.54	643	6.15	4023	1.87
kro124p	100	Not	Not	3600.00	Not	3600.00	Not	3600.00
ftv100	101	13	371385	92.20	1643	21.87	54688	30.93
ftv110	111	3	583133	160.44	6313	112.03	79045	89.56
ftv120	121	31	3892497	1088.57	13721	305.71	137563	327.31
ftv130	131	32	256855	95.71	715	19.01	11269	28.52
ftv140	141	8	78951	42.25	1683	55.82	12667	13.30
ftv150	151	114	15437	9.18	625	26.37	3635	4.67
ftv160	161	72	Not	3600.00	Not	3600.00	330458	496.92
ftv170	171	2321	1796149	1299.34	Not	3600.00	412059	656.59
rbg323	323	0	3	0.00	1	0.49	1	0.05
rbg358	358	0	3	0.05	1	0.55	1	0.00
rbg403	403	0	3	0.05	1	0.82	1	0.05
rbg443	443	0	3	0.11	3	3.02	2	0.05
Unsolved		4		3		4		2
Average		10.00	445028	73.94	9141	35.02	67211	26.27
Median		1.00	25831	3.24	1683	7.20	4756	1.87

Firstly, we compare our DFS algorithms with the CDT benchmark in Tables 3.7, 3.8, and 3.9. Although the median solution times of CDT are generally

Table 3.8. Average search tree sizes and solution times of asymmetric random instances

	CDT	$BnB(Cost)$		$BnB(ECS)$		$BnB(SCS)$	
n	Time	Tree	Time	Tree	Time	Tree	Time
60	0.00	380.8	0.03	32.3	0.12	22.1	0.02
70	0.00	452.8	0.04	31.2	0.18	24.7	0.03
80	0.10	901.4	0.13	21.7	0.24	25.6	0.07
100	0.10	900.2	0.19	17.4	0.26	13.5	0.05
200	0.10	3639.0	3.30	85.8	7.30	79.6	0.84
300	0.30	17849.8	48.10	93.6	28.70	150.6	5.34
400	0.20	28499.4	141.00	74.2	54.10	121.6	11.38
500	0.40	43457.6	368.70	187.8	268.40	225.3	35.98
1000	1.50	92289.0	3951.60	142.1	1556.90	373.9	157.23

Table 3.9. Average search tree sizes and solution times of symmetric and sparse instances

		CDT	BnB(Cost)		BnB(ECS)		BnB(SCS)	
Instance	n	Time	Tree	Time	Tree	Time	Tree	Time
Degree of sparsity 50%	100-400	0.6	12291	44.7	90	39.1	60	2.33
Degree of sparsity 75%	100-400	0.7	12882	48.9	109	41.57	81	3.90
Degree of sparsity 90%	100-400	0.7	14109	55.63	116	63.63	84	4.70
Degree of symmetry 0.33	60-80	0.1	1963	0.27	195	0.86	139	0.10
Degree of symmetry 0.66	60-80	0.5	6763	1.08	2193	9.63	1097	0.62
Full symmetry	60-80	[1]	446335	58.63	30347	116.73	379412	121.03

Table 3.10. Search tree sizes and solution times of Buriol instances, $k' = 5$

		CDT	$BnB(Cost)$		$BnB(ECS)$		$BnB(SCS)$	
Instance	n	Time	Tree	Time	Tree	Time	Tree	Time
ulysses16	16	1	10165	0.11	4773	0.27	2337	0.05
ulysses22	22	45	857027	10.27	209765	24.56	189735	7.36
gr24	24	0	1329	0.05	401	0.16	371	0.05
fri26	26	0	12725	0.22	697	0.27	1943	0.11
bayg29	29	3600	6555	0.16	29939	49.34	5317	0.33
bays29	29	0	13931	0.33	2037	0.66	4346	0.33
gr48	48	0	3331593	171.21	155	0.00	196635	25.38
att48	48	1252	63782833	2815.93	439781	543.19	120297	16.92
eil51	51	1	1193015	66.54	95403	138.63	10167	1.37
pr76	76	3600	Not	3600.00	Not	3600.11	Not	3600.05
eil76	76	3600	8772639	1058.55	9013	48.41	652530	210.93
gr96	96	3600	Not	3600.00	Not	3600.05	Not	3600.05
Unsolved		4		2		2		2
Average		162.38	8650327	383.08	94127	88.47	65729	6.45
Median		0.50	435479	5.30	3405	0.47	7257	0.85

Table 3.11. Search tree sizes and solution times of Buriol instances, $k' = 50$

		CDT	$BnB(Cost)$		$BnB(ECS)$		$BnB(SCS)$	
Instance	n	Time	Tree	Time	Tree	Time	Tree	Time
ulysses16	16	0	2481	0.00	135	0.00	181	0.00
ulysses22	22	0	35577	0.38	4270	0.71	4845	0.22
att48	48	0	147201	6.65	19	0.50	37	0.00
bayg29	29	0	177	0.00	7	0.00	7	0.00
bays29	29	0	323	0.00	153	0.50	23	0.00
eil51	51	1	16869	0.93	9	0.50	13	0.00
fri26	26	0	2481	0.00	1014	0.33	33	0.00
gr24	24	0	49	0.00	9	0.00	19	0.00
gr48	48	0	4503	0.27	43	0.50	265	0.50
pr76	76	0	1643	0.27	424	3.80	91	0.50
eil76	76	0	243	0.00	9	0.50	13	0.00
gr96	96	0	851473	218.19	59613	763.90	403	0.44
kroA100	100	0	21282	38.19	95	1.32	82	0.05
kroD100	100	0	21294	0.11	201	2.36	98	0.05
rd100	100	1	7910	38.02	25	0.33	434	0.22
eil101	101	0	629	0.05	3	0.05	21	0.00
lin105	105	0	14379	42.47	17	0.27	15	0.00
ch130	130	3600	Not	3600.00	38221	969.73	Not	3600.00
ch150	150	3600	Not	3600.00	Not	3600.00	Not	3600.00
Unsolved		2		2		1		2
Average		0.12	66383	20.33	3885	45.62	387	0.12
Median		0.00	4503	0.27	69	0.50	37	0.00

the shortest for the ATSPLIB and the Buriol instances, CDT solves the smallest number instances. The CDT algorithm solves randomly generated and small ATSPLIB instances in very short solution times compared to our DFS algorithms. However, the CDT algorithm has more difficulties with the harder and larger ATSPLIB instances, and with symmetric and Buriol instances. The poor performance of the CDT algorithm for difficult problem instances can be explained as follows. Our DFS algorithms use the AP solution of the parent subproblem to speed up the solution of the current subproblem. A BnB algorithm with BFS strategy, on the other hand, proceeds more or less randomly through the search tree, so that it is too memory-intensive to use the AP solution of the parent subproblem. Carpaneto *et al.* (1995) try to overcome this problem by using the AP solution of the top node of the search tree as a starting solution. The AP solutions in the beginning of the BnB process are computed quickly, in $O(n^2)$ time, since these APs are similar to the initial AP at the root node of the search tree. However, when a large number of subproblems is expanded, the differences between the current AP and the AP at the top node grow. The solution time needed to solve these APs approach $O(n^3)$.

If we compare our results to the other algorithms in Fischetti *et al.* (2002), the FT-add method is slightly faster for many instances. This algorithm also applies relaxations different from the AP. As a consequence, it is able to keep search trees small for most instances, and keep the computational overhead within reasonable limits. Unfortunately, a direct comparison on the same computer cannot be made, since, to the best of our knowledge, no online source code is available. The Concorde solver and the FT Branch and Cut solver in Fischetti *et al.* (2002) both run under C-Plex. Both algorithms appear to be slower than our algorithms for randomly generated instances, but faster for the instances from the TSP library.

In a comparison among the DFS algorithms, Tables 3.7, 3.8, and 3.9 show that $BnB(SCS)$ obtains the smallest solution times for many instances, sparse instances, and instances with degree of symmetry 0.33 and 0.66, but the solution times of $BnB(Cost)$ are slightly better for fully symmetric instances. For sparse instances, the solution times of $BnB(SCS)$ are also shorter, but the advantage reduces as sparsity increases. $BnB(ECS)$, using the ECS lower bound and branching rule, generally requires too much tolerance calculation time to be competitive, in spite of its small search trees. In Table 3.10 and Table 3.11, $BnB(SCS)$ displays the fastest solution times, even though the instances with $k' = 5\%$ are almost symmetric.

Finally, we analyze the source of the search tree reductions of $BnB(ECS)$ and $BnB(SCS)$. A BnB algorithm first finds an optimal solution and then proves the optimality of that solution, i.e., all remaining subproblems are discarded. Table 3.12 shows that $BnB(Cost)$ spends a relatively large amount of time on finding an optimal solution compared to $BnB(ECS)$ and $BnB(SCS)$, particularly for non-symmetric random instances. This result indicates that the improved branching of tolerance-based algorithms is the predominant source of the search tree reductions. Table 3.12 also indicates that fast algorithms often spend the smallest fraction of time on finding an optimal solution. The value of an optimal solution is the tightest possible upper bound and can be used to fathom a large number of subproblems. We conclude that accurate branching is the key to good performance of Depth First Search BnB algorithms. Moreover, since tolerance-based algorithms find optimal solutions faster, the results in case of premature termination are likely to be better than for cost-based algorithms.

Table 3.12. Percentage of subproblems solved before an optimal solution is found

Instance	$BnB(Cost)$	$BnB(ECS)$	$BnB(SCS)$
ATSPLIB	30.48%	64.48%	39.63%
Degree of symmetry 0.33	92.60%	62.59%	70.42%
Degree of symmetry 0.66	85.12%	49.14%	61.65%
Full symmetry	35.70%	31.71%	32.08%
Asymmetric random	90.02%	77.27%	75.76%
Degree of sparsity 50%	95.15%	82.49%	76.88%
Degree of sparsity 75%	96.56%	80.77%	77.65%
Degree of sparsity 90%	95.83%	72.30%	82.35%
Buriol, $k' = 5$	82.57%	62.25%	66.16%
Buriol, $k' = 50$	97.27%	52.47%	41.93%

This property may be very useful in case of solving large problems within limited times; see Zhang (1993).

There seems to exist the following paradox. The use of *lower bounds* based on tolerances cause the largest search tree reductions according to Goldengorin *et al.* (2004), and hence, one may expect that these algorithms need less time to prove the optimality of the solution at hand. However, Table 3.12 points out that tolerance-based BnB algorithms need relatively less time to find an optimal solution. An explanation is that the new lower bounds cut off a large number of subproblems *before* an optimal solution is found.

3.10 Summary and Future Research Directions

We presented an experimental analysis of tolerance-based BnB type algorithms for the ATSP, and compared it with cost-based BnB algorithms. Tolerance-based algorithms reduce the search tree sizes substantially. For random instances, including both instances with degree of symmetry 0.33 and 0.66, and sparse instances, the computation times are substantially better.

The better performance of tolerance-based BnB algorithms is mainly caused by improved *branching*: a better choice of entries to be included and excluded. Upper tolerances provide better predictions of which arcs in a relaxation solution should be preserved, the *survival set*, and which arcs should be deleted, the *extinction set*.

We apply the concept of *offenders*: sources of infeasibility which must be removed from a relaxation solution in order to obtain a feasible solution for the original hard problem. The minimal cost of removing such an offender can

be determined using tolerance values. This idea is used to construct new lower bounds, but it also enables the BnB algorithm to make branching steps with large increases in solution value without "jumping" across an optimal ATSP solution. The largest increase in lower bound is obtained if we consider all offenders, the *Entire Cycle Set*, which has of course the drawback of very long tolerance calculation times. It is shown that a good approximation is the *Smallest Cycle Set*, which only uses a smallest cycle in the set of offenders, so that only a few tolerances need to be calculated. Branching on the smallest cycle is a good choice, not only because a small number of new subproblems is generated, but also because it is very likely that a large branching step towards an optimal ATSP solution is made.

Tolerance-based BnB algorithms have one major drawback: they have to compute tolerances at every node of the search tree. In spite of this drawback, it turns out that the BnB algorithm with the Smallest Cycle Set bound and branching rule usually require shorter solution times than cost-based BnB algorithms. This algorithm is faster for difficult instances than the state-of-the-art BnB algorithm from Carpaneto *et al.* (1995).

The idea of branching on tolerances can be seen as similar to the idea of *strong branching* in Integer Programming; see Achterberg *et al.* (2004). Strong branching first explores the additional cost of setting a fractional variable at an integer value, and then decides on which variable to branch. Similar to 0tolerance-based branching, it first computes the additional cost of removing infeasibilities, the fractional value of a variable, before it takes the branching step. In Achterberg *et al.* (2004), it is found that strong branching should be done only at specific nodes of the search tree. Similar strategies can be developed for tolerance-based algorithms.

An interesting direction of research is to develop book-keeping techniques that accelerate tolerances computations, and lead to solution time reductions for ATSP instances. Another direction of research is to construct tolerance-based algorithms for other COPs. Preliminary experiments with these algorithms are very promising as well.

Chapter 4

Additional Topics on Tolerance-Based Algorithms

4.1 Introduction

In Chapter 3, tolerance-based Branch and Bound (BnB) algorithms are introduced for the Asymmetric Traveling Salesman Problem (ATSP). The performance of these algorithms is compared to cost-based algorithms and BnB methods from the literature. Tolerance-based BnB methods prove to be effective for various instances of the ATSP, but not for all. In this chapter, we try to find instances for which tolerance-based BnB is most effective, and try to find reasons for it being so. Moreover, we suggest improvements to tolerance-based BnB algorithms, such as increased lower bounds.

First, we consider the theory of *complexity transitions*, explained in Hogg *et al.* (1996). The difficulty of a randomly generated instance of a COP often depends on the values of the parameters describing the distribution from which the random instance is drawn. Take for example the ATSP, introduced in Section 2.1. An instance of the ATSP is defined by the cost matrix C containing the distances between each pair of locations. Assume that each entry in the matrix C is drawn from a uniform distribution supported on $\{1, \ldots, R\}$. These randomly generated instances are more difficult to solve if the range of the distribution R is large; see Zhang and Korf (1996). In Section 4.2, we compare the performance of tolerance-based BnB algorithms with that of cost-based BnB for various values of R.

In Section 4.3, we consider alternative branching rules for the ATSP. In Section 3.5, we try to predict which arcs from an Assignment Problem (AP) solution are also in some optimal solution of the ATSP instance with the same cost matrix C. The results indicate that the predictions based on upper tolerance values are more accurate than those based on cost values. We restrict ourselves Depth First BnB algorithms. An inaccurate prediction at a node close to the root of the branch and bound tree would cause the algorithm to fruitlessly search through a large number of subproblems; see Section 3.5. It may be worthwhile to restrict the tolerance computations to a small part of the search tree, since these computations may take much time. Branching on tolerances only at top nodes of the search tree seems to make sense, but is this true? We compare *hybrid algorithms*, using tolerances at selected nodes and costs elsewhere, to the usual cost-based and tolerance-based algorithms from Section 3.7.

Finally, in Section 4.4, we try to improve tolerance-based lower bounds for problems that have the Minimum Spanning Tree Problem (MSTP) as a relaxation. The lower bounds from Section 3.6 are formed by adding the upper tolerance value of a single arc, even when there is a large number of subcycles that should be broken. When the number of offenders is large, one is inclined to remove multiple offenders simultaneously. We develop new lower bounds for the Degree-Constrained Minimum Spanning Tree Problem (DCMSTP) for which we use the regular Minimum Spanning Tree Problem (MSTP) as a relaxation. This lower bound is obtained by adding the upper tolerance values of specific edges.

4.2 Branch and Bound and Complexity Transitions

The fact that a problem is $\mathcal{NP}$-hard does not imply automatically that all very large instances of the problem are unsolvable. There exist conditions under which $\mathcal{NP}$-hard problem instances are actually polynomially solvable; see, for example, Burkard (1997). In this section, we consider problem instances that are *on average* polynomially solvable. This means the following. Suppose that instances of a COP are drawn from a certain distribution with prespecified parameter values. In case of the ATSP, we assume that the intercity distances in C are drawn from a uniform distribution supported on $\{1, \ldots, 1000\}$. In Section 1.3,

a definition of average case complexity is given. What is, on average, the complexity of such randomly generated ATSP instances?

In this section, we compare two algorithms from Chapter 3, namely $BnB(Cost)$ and $BnB(SCS)$ for random instances with different average complexities. Recall that $BnB(SCS)$ branches on an arc in the smallest cycle with the lowest upper tolerance value, and that $BnB(Cost)$ branches on an arc in the smallest cycle with the highest cost value. The complexity of the instances is artificially varied using the theory of *complexity transitions*.

Complexity transitions analysis is a statistical analysis of the complexity of $\mathcal{NP}$-hard problems (Hogg, 1996). Complexity transitions are special cases of *phase transitions*. According to the definition from Wilson (1979), a phase transition is a dramatic change of some system property; its 'state' crosses a certain virtual boundary when certain control parameters change. Phase transitions are usually studied in physical systems. An example is the melting of a solid substance (Hogg *et al.*, 1996), for which the parameter under control is the environment temperature and the output is the state of the substance. A small increase in the environment temperature across the melting temperature leads to an abrupt softening of the substance. This is a phase transition, since the transition takes place abruptly and not smoothly. A phase transition is schematically depicted in Figure 4.1. The state of the system in this figure depends on the values of two parameters. The curved line marks the border between the two states 1 and 2, for example, the state in which the substance is solid and substance is fluid.

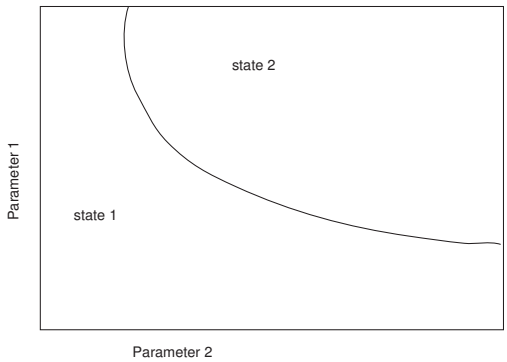

Figure 4.1. Phase transition

The study of *complexity transitions* considers a search procedure for finding

a solution of an $\mathcal{NP}$-hard problem instance as a system (Hogg *et al.*, 1996). An example of such a search procedure is Branch and Bound (Zhang and Pemberton, 1994). The input of the 'system' consists of the instance to be solved; the output is the number of nodes in the search tree visited before the instance is solved. It is assumed that the parameter choice of the random distribution from which the instances are drawn determines the complexity of the instances. Complexity theory results show that there is a thin (virtual) border between instances with search trees that are, on average, polynomial in the input size, the easy instances, and instances with exponential search trees, the hard instances. For an explanation of the terms 'polynomial' and 'exponential', we refer to Section 1.3. The purpose of the analysis is to find combinations of parameter values which mark the border between easy and hard instances. The 'abrupt' transition from easy to hard instances is called a *complexity transition*. It appears for many COPs; see, for example, Dunne *et al.* (2000) and Van der Veen (1992).

The importance of this type of complexity analysis is twofold. Firstly, it is helpful for instance generation in computational experiments. A possible pitfall in a computational experiment is that algorithms are only tested on instances with average polynomial complexity, which may lead to results that are invalid for more difficult problem instances (Johnson, 1991). Complexity analysis can be used to generate both easy and hard random instances. Secondly, since the border between easy and difficult instances is typically thin, instances situated on the exponential side of the transition may be transferred to the polynomial side by means of minor changes in the data; see Zhang and Pemberton (1994). The so-called Data Correcting Algorithms from Goldengorin (2002) work along a similar principle.

In this section, we investigate complexity transitions for the Asymmetric Traveling Salesman Problem (ATSP). Previous studies on complexity transitions for the ATSP are Zhang and Korf (1996), Zhang (2003), and Zhang (2004). In these studies, random instances drawn from both uniform and lognormal distributions are considered; we only take the uniform distribution into account here. The uniform distribution is supported on $\{1, \ldots, R\}$, where the parameter under control is R, the *range* of the distribution. Theoretical and computational results in the papers mentioned above show that search tree sizes undergo a complexity transition as the value of R changes.

We perform computational experiments on a cost-based BnB algorithm as well as on a tolerance based algorithm: $BnB(SCS)$ and $BnB(Cost)$ from Chapter 3, respectively. All instances are of size 100, and the values of c_{ij} are drawn from a uniform distribution supported on $\{1, \ldots, R\}$. The value of R runs from 10 to 10,000 in varying step sizes. One hundred instances are generated and solved for each value of R. The results are presented in Table 4.1. We compare both the average search trees and the average solution times for each algorithm and for each selected value of R. (Zhang, 1993) have already established that complexity transactions exist for the ATSP when R is the input parameter. The question is: do cost-based and tolerance-based BnB algorithms behave similarly for instances on both sides and in the complexity transition.

Table 4.1. Average search trees and solution times of $BnB(SCS)$ and $BnB(Cost)$

Range R	$BnB(SCS)$		$BnB(Cost)$	
	Search tree	Solution time	Search tree	Solution time
10	1.40	0.001	3.34	0.001
25	6.94	0.010	10.64	0.004
50	16.04	0.033	114.04	0.028
100	27.66	0.053	391.90	0.075
200	34.70	0.077	638.74	0.124
300	31.27	0.070	810.22	0.158
400	32.62	0.071	861.42	0.162
500	31.04	0.078	813.04	0.157
700	36.90	0.085	884.94	0.185
1000	34.22	0.074	1120.02	0.219
2000	39.10	0.101	1036.68	0.209
5000	37.76	0.100	1090.22	0.220
10000	34.58	0.125	995.98	0.209

Figure 4.2 presents the search tree sizes of $BnB(Cost)$ and $BnB(SCS)$ graphically. In order to make the patterns of both algorithms comparable, we set the search trees for $R = 100$ equal to 100% and compute the remaining search tree sizes relative to $R = 100$. So for $BnB(Cost)$, the relative search tree of $R = 1000$ is $\frac{1120.02}{391.90} \times 100\% = 28.58\%$.

Figure 4.2 also shows that the search tree sizes increase strongly between $R = 25$ and $R = 200$. For $R = 10$ and 25, the BnB systems are in the 'easy'

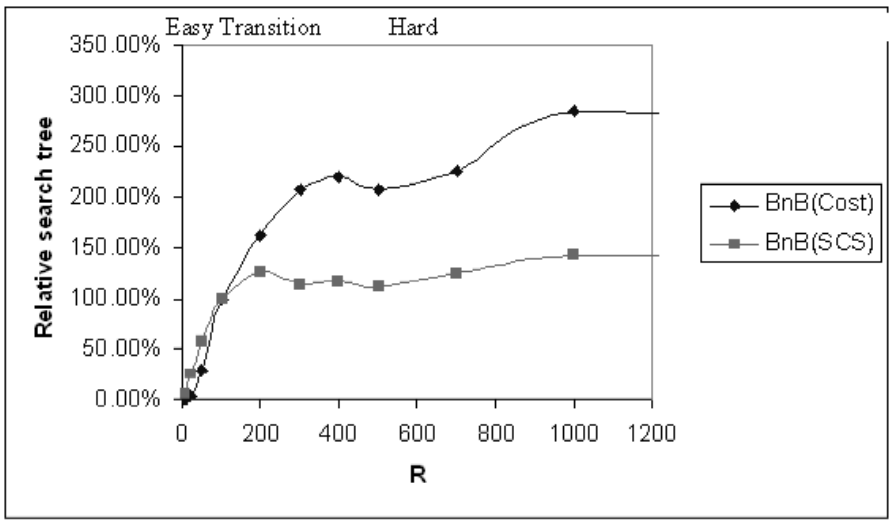

Figure 4.2. Relative search trees for $n = 100$

.

state: instances are solved quickly. When R reaches 200, the search trees and solution times are relatively large, but when R is increased beyond that, only small changes can be identified. This is the 'hard' state. The region between $R = 25$ and $R = 200$ is the *complexity transition* region. From Figure 4.2, we expect a sudden increase in search tree sizes, but the borderline is not so sharp. However, the pattern is similar to the one found in Zhang (1993).

The increase for $BnB(Cost)$ is much stronger than the increase for $BnB(SCS)$. For small values of R, the search trees of $BnB(Cost)$ and $BnB(SCS)$ are almost equally large. It appears that for small values of R, branching on tolerances does not make too much sense. Most tolerance values are equal to 0, and branching on tolerances is now nothing more than random branching. In contrast, when the value of R is large, the trees of $BnB(SCS)$ are clearly smaller than those of $BnB(Cost)$. So now it makes much more sense to branch on tolerances. Our explanation is that there are many different intercity distances, so that low cost arcs are often not part of an optimal ATSP solution. Section 3.5 shows, on the other hand, that arcs with high upper tolerances frequently belong optimal ATSP solutions.

To summarize the results of this section, it appears that the tolerance-based

BnB algorithm $BnB(SCS)$ is more powerful on relatively difficult randomly generated ATSP instances than the cost-based $BnB(Cost)$.

4.3 Hybrid Branch and Bound Algorithms

One of the main disadvantages of using the relaxation tolerances in an algorithm is the time required to compute the tolerance values. In this section, we try to restrict the tolerance computations to specific nodes of the search tree. To that end, we introduce *hybrid BnB algorithms*, which apply tolerance-based branching and bounding only at nodes where it is likely to be effective and cost-based branching and bounding at other nodes.

Which nodes in the search tree are good candidates for branching on tolerances? We have seen in Section 3.5 that, at the top nodes of the search tree, it is important to include or exclude the correct arcs. If an arc from all optimal ATSP tours is excluded in the top node, the algorithm searches through a large part of the solution space where no good ATSP solution can be found. If, on the other hand, a wrong arc is included or excluded in a bottom node of the search tree, then the consequence of this mistake is that the BnB only searches through a few additional nodes in the search tree in vain. Following this line of reasoning, it appears to be so that tolerance-based branching is effective at top nodes of the search tree, and at nodes close to the top node. Note that when upper tolerances are available for branching, they are automatically available for bounding; see Section 3.6.

We define the *depth* of a node in the search tree as the number of edges on the shortest path from that node to the top node in the search tree; see Section 2.2. We hypothesize that tolerance-based branching and bounding is most effective at small depths in the search tree, i.e., close to the top node. Let δ be a user-defined parameter. We introduce the set of hybrid BnB algorithms $HBnB(\delta)$ branch on upper tolerances if the depth of the current node is smaller than a user-defined parameter δ; branching and bounding based on costs is performed at the other nodes. If $\delta = 0$, then the algorithm is the fully cost-based $BnB(Cost)$, and if $\delta \geq n^2$, we obtain the fully tolerance-based $BnB(SCS)$. A graphical example of a hybrid BnB algorithm ($\delta = 2$) is depicted in Figure 4.3.

Another potential advantage of the choice of an intermediate value of δ may

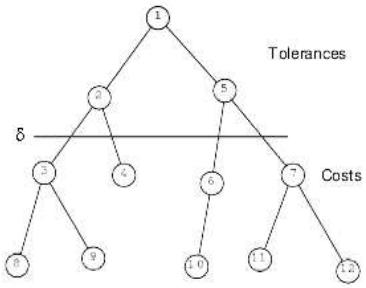

Figure 4.3. Hybrid BnB search tree ($\delta = 2$)

be the following. Since he SCS branching rule breaks a smallest cycle, the time needed to determine the SCS arc is proportional to the cardinality of the smallest cycle; see Section 3.6. When the BnB algorithm proceeds deeper into the search tree, the remaining cycles tend to be larger. Hence, the complexity of computing the SCS arc deep in the search tree is likely to be relatively high.

Computational experiments are conducted on some practical instances from the ATSPLIB (Reinelt, 1995). The value of δ is varied. For each value of δ and for each instance, search tree sizes and solution times are reported in the figures below. The tables containing the results, Table 4.4 and Table 4.3, can be found in the Appendix.

Figure 4.3 present the search tree sizes of the selected ATSPLIB instances. If applying upper tolerance-based BnB is indeed most effective at top nodes, a large decrease in the search tree size can be expected between $\delta = 0$ and $\delta = 2$ and $\delta = 5$. However, it turns out that for more than half of the tested instances, the search trees *increase* in size, when δ increases from 0 to 2. The same holds when δ is increased from 2 to 5.

Figure 4.3 present the solution times. The results indicate that the smallest solution times are often achieved by either the fully cost-based or the fully

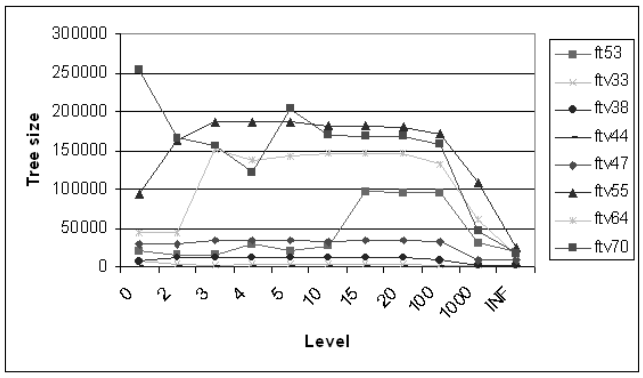

Figure 4.4. Search tree sizes of ATSPLIB instances

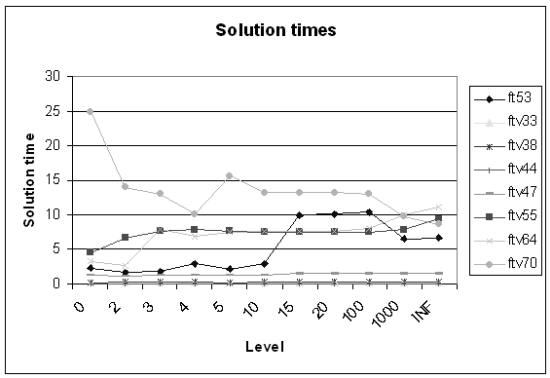

Figure 4.5. Solution times of ATSPLIB instances

tolerance-based algorithm, depending on the instance at hand. The experiments also indicate that the longest solution times are often achieved by either $BnB(SCS)$ ($\delta = \infty$) or $BnB(Cost)$ ($\delta = 0$) as well. In particular when we take $\delta = 2$, solution times are reasonable if either $BnB(Cost)$ or $BnB(SCS)$ requires relatively long solution times.

The computational experiments indicate that the largest reductions are usually not achieved for small values of δ. This finding implies that branching and bounding on upper tolerances is not only effective at top nodes of the BnB search tree, but that it should be applied at each node in the search tree. However, more extensive research is needed to provide conclusive answers. For example, it is interesting to consider randomly generated instances as well.

An interesting direction of future research is to perform *reduced tolerance*

computations, meaning that tolerance computations are performed only partially. In Chapter 3, we have already observed that there is a large degree of similarity between tolerance-based BnB algorithms for COPs and strong branching for Integer Linear Programming (ILP) problems. The *strong branching approach*, also called *pseudo-cost based approach* in Linderoth and Savelsbergh (1997), solves ILP problems as follows; see Achterberg *et al.* (2004). Assume that we use BnB to solve an ILP problem, with the usual LP relaxation. For any node k in the search tree, let X_k be the set of integer variables with fractional values, and let $x \in X_k$. If the value of a variable $x = f$, then the LP problem P_1 with the additional constraint $x \geq \lceil f \rceil$ and the problem P_2 with the constraint $x \leq \lfloor f \rfloor$ are solved. The *pseudo-cost* of the variable x is a weighted average of the increase in objective function of optimal solutions of P_1 and P_2, the weights being $\frac{1}{1-(f-\lfloor f \rfloor)}$ and $\frac{1}{f-\lfloor f \rfloor}$, respectively. The variable with the largest pseudo-cost is selected as the next branching variable. Both pseudo-cost based and tolerance based branching rules explore the objective value of the solution obtained after using each branching variable before the actual branching is performed.

Linderoth and Savelsbergh (1997) report that the use of pseudo-costs in branching reduces search trees substantially. However, the pseudo-cost computations take too much time, even though search tree reductions are large. In Linderoth and Savelsbergh (1997), it is found that reduced pseudo-cost computations lead to the smallest solution times. In reduced pseudo-cost computations, the LP-problems P_1 and P_2 are not fully solved, but only to a prespecified number of iterations of the LP-solver. A similar strategy can be followed with tolerances.

4.4 An Additive Tolerance-Based Lower Bound for the DCMSTP

In Section 3.7, upper tolerance-based lower bounds are introduced for the Asymmetric Traveling Salesman Problem (ATSP), and its Assignment Problem (AP) relaxation. These lower bounds use the concept of *offenders*, sources of infeasibility in the current relaxation solution, to construct a lower bound. The minimum cost of removing one offender is added to the lower bound. Can we also take the cost of removing more than one offender at once and still guarantee a

lower bound?

An AP solution is infeasible for the ATSP if it consists of more than one subcycle, so subcycles are offenders. The AP solution value itself is a lower bound for the ATSP, but in order to obtain an ATSP solution, at least one subcycle should be "broken". It is shown in Section 3.7, that the cost of breaking a subcycle is at least the minimum value across all upper tolerance values of arcs in that subcycle, called the *bottleneck tolerance*. This value is then added to the the optimal AP solution value in order to obtain a lower bound for optimal ATSP solution of that instance. For randomly generated instances, these tolerance-based lower bounds appear to be a good approximation of the value of optimal ATSP solutions, but for most practical instances, the gap between the lower bound and the value of the optimal solution is still wide.

When there are several subcycles in the AP solution, the lower bounds would be much tighter when the sum of the bottleneck tolerances of a set of subcycles is added simultaneously to the value of the AP solution. Unfortunately, we cannot guarantee that the value obtained is a lower bound for an ATSP solution. Consider the counterexample depicted in Figure 4.6. The number next to an arc denotes the cost of the arc. Arcs with cost 0 form the (unique) AP solution of this example, consisting of four subcycles. Consider the subcycle $(\{(1,2),(2,1)\}$. The bottleneck tolerance value is 4, achieved on the arc $(2,1)$. The cheapest AP solution after the deletion of the arc $(2,1)$ consists of an ATSP tour through all locations with cost 4; this is the unique shortest ATSP tour. The bottleneck tolerance values of the other cycles are 4 as well. So if the bottleneck tolerance of another subcycle is added as well, say $\{(3,4),(4,5),(5,3)\}$, then the value of the suggested lower bound is $0+4+4=8$, whereas the value of the optimal AP solution is 4. Even though there are four cycles, the bottleneck tolerance of only one subcycle can be added to the lower bound.

In case of the AP, it is generally not possible to add the upper tolerances from more than one offender at once and still guarantee a lower bound. Fortunately, this negative result for the AP does not extend to all COPs. We show in this section that the sum of the upper tolerances of more than one element can be added to the lower bound for problems which have the *Minimum Spanning Tree Problem* (MSTP) as a relaxation.

Given an undirected graph G with edge set $\mathcal{E}$ and vertex set V, the MSTP is

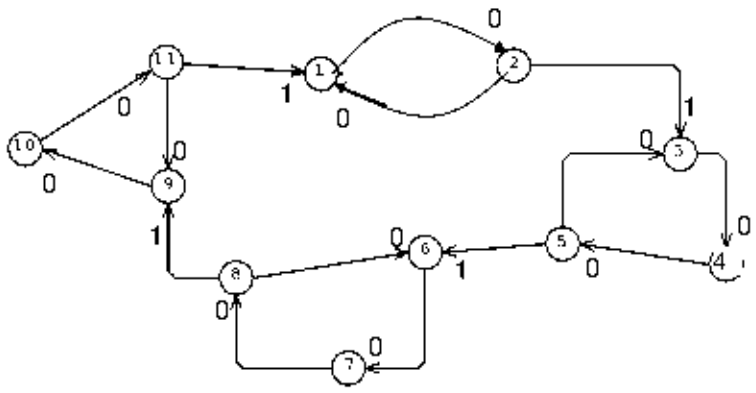

Figure 4.6. Counterexample: Removal of the arc $(2, 1)$ leads to an ATSP tour

the problem of connecting n vertices in V by means of $n-1$ edges from $\mathcal{E}$ against minimum cost. The resulting solution is called a *Minimum Spanning Tree (MST)*. The MSTP can be solved in $O(n^2)$ time with, among others, Prim's algorithm (Prim, 1957). The MSTP may serve as a relaxation of the Symmetric Traveling Salesman Problem (STSP), and of the *Degree-Constrained Minimum Spanning Tree Problem (DCMSTP)*; see for example Volgenant (1989). Let $d \geq 2$. The d-DCMSTP is the problem of finding an MST with at most d edges incident to each vertex. Applications of the d-DCMSTP are found mainly in communication networks; see Krishnamoorthy *et al.* (2001) and Caccetta and Hill (2001). The d-DCMSTP is $\mathcal{NP}$-hard for any value of $d \geq 2$ (Garey and Johnson, 1979).

In the framework of Section 3.3, the MSTP can be denoted by $(\mathcal{E}, \mathcal{T}, C, f_C)$. The set $\mathcal{E}$ denotes the set of edges, the set $\mathcal{T}$ denotes the set of spanning trees. Let n be the number of vertices to be connected. Each $T \in \mathcal{T}$ has the following well-known properties: it contains exactly $n - 1$ edges, contains no subcycles, and is connected. C refers to the *instance* of the MSTP, the cost of an edge $e \in \mathcal{E}$ is denoted by $c(e)$, and f_C denotes the cost function, i.e., $f_C(T)$ is the sum of the costs of all edges in $T \in \mathcal{T}$. We denote the set of MSTs by $\mathcal{T}^*$.

Let $E \subseteq \mathcal{E}$. Define $\mathcal{T}_{-}(E) = \{T \in T : T \cap E = \emptyset\}$, and let $\mathcal{T}^*_{-}(E)$ be the set of MSTs in $\mathcal{T}_{-}(E)$. Clearly, $\mathcal{T}_{-}(\emptyset) = \mathcal{T}$ and $\mathcal{T}^*_{-}(\emptyset) = \mathcal{T}^*$. When we consider the

set of edges $\{e_1, \ldots, e_k\}$, we write $\mathcal{T}_-(e_1, \ldots, e_k)$ and $\mathcal{T}^*_-(e, \ldots, e_k)$ instead of $\mathcal{T}_-(\{e_1, \ldots, e_k\})$ and $\mathcal{T}^*_-(\{e_1, \ldots, e_k\})$, respectively. Take any $T^* \in \mathcal{T}^*$. Since the MSTP is a monotonous COP (see Section 3.3), the upper tolerance $u_{T^*}(e)$ of any arc $e \in T^*$ can be computed as follows (see e.g. Goldengorin *et al.* (2006)):

$$u_{T^*}(e) = f_C[T^*_-(e)] - f_C[T^*]. \tag{4.4.1}$$

The following lemmas include convenient characteristics of upper tolerances of MSTP solutions.

Lemma 4.4.1. *Let $T \in \mathcal{T}^*$ and let $e \in T$. Moreover, let $\mathcal{T}_-(e) \neq \emptyset$. There is $T_- \in \mathcal{T}^*_-(e)$ such that $|T \oplus T_-| = 2$.*[1]

Lemma 4.4.1 is proven in, among others, Helsgaun (2000). Let $T_1, T_2 \in \mathcal{T}$, $e \in T_1$ and $\mathcal{T}_-(e) \neq \emptyset$. T_2 is called a *maximal intersection* tree of T_1 w.r.t. e. We define $MIT(T_2, e)$ as follows: $T_2 \in MIT(T_2, e)$, iff $|T_1 \oplus T_2| = 2$. Moreover, for each edge e, we denote and define the substitute edge of e as $p_{T_1}(e) \in T_2 \backslash T_1$. Clearly, $T_1 \oplus T_2 = \{e, p_{T_1}(e)\}$.

When $T^*_-(e)$ is a maximal intersection tree of T^*, it holds that $T^* \oplus T^*_-(e) = \{e, p_{T^*}(e)\}$. It follows from (4.4.1) that $u_{T^*}(e) = f_C[T^*_-(e)] - f_C[T^*] = f_C[T^*] + c[p_{T^*}(e)] - c(e) - f_C[T^*] = c[p_{T^*}(e)] - c(e)$.

Lemma 4.4.2. *Let $e \in T^* \in \mathcal{T}^*$, and let $T^*_-(e) \in \mathcal{T}^*_-(e) \cap MIT(T^*, e)$. Let $a \in T^* \cap T^*_-(e)$ with $\mathcal{T}_-(a, e) \neq \emptyset$. It then holds that $u_{T^*_-(e)}(a) \geq u_{T^*}(a)$.*

Proof. By definition, $p_{T^*}(a) = T^*_-(a) \setminus T^*$ for any maximal intersection tree $T^*_-(a)$ w.r.t. $a \in T^*$. To prove the lemma, we use Lemma 4.4.1 and distinguish between the following two cases.

Case 1: $p_{T^*}(a) \neq p_{T^*}(e)$. Then $u_{T^*_-(e)}(a) = f_C[T^*_-(a, e)] - f_C[T^*_-(e)] = c[p_{T^*}(a)] - c[a] = (f_C[T^*] + c[p_{T^*}(a)] - c[a]) - f_C[T^*] = f_C[T^*_-(a)] - f_C[T^*] = u_{T^*}(a)$. Hence, $u_{T^*_-(e)}(a) = u_{T^*}(a)$.

Case 2: $p_{T^*}(a) = p_{T^*}(e)$. From Lemma 4.4.1, it follows that $\exists T^*_-(e) \in \mathcal{T}^*_-(e)$, such that $T^*_-(e) := T^* \backslash \{e\} \cup \{p_{T^*}(e)\}$. Since $p_{T^*}(a) = p_{T^*}(e), p_{T^*}(a) \in T^*_-(e)$. As a consequence, $T^*_-(e) \cup \{p_{T^*}(a)\} \setminus \{a\} \notin \mathcal{T}$, since it contains

[1] $\oplus$ denotes the symmetric difference, $A \oplus B = (A \cup B) \setminus (A \cap B)$.

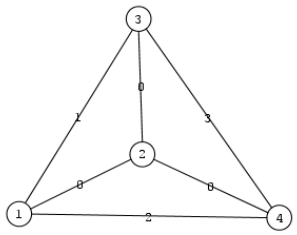

Figure 4.7. MSTP example

only $n-2$ edges. So a obtains a new substitute edge $p_{T^*_-(e)}(a) \neq p_{T^*}(a)$ It holds that $p_{T^*_-(e)}(a) = T^*_-(a,e) \setminus T^*_-(e)$ for the maximal intersection tree $T^*_-(a,e) \in \mathcal{T}^*_-(a,e)$ of $T^*_-(a)$ w.r.t. a. We now prove by contradiction that $c[p_{T^*_-(e)}(a)] \geq c[p_{T^*}(a)]$.

Assume that $c[p_{T^*_-(e)}(a)] < c[p_{T^*}(a)]$. Let $T^*_-(a) \in \mathcal{T}^*_-(a)$. Let $T^*_- := T^* \setminus \{a\} \cup \{p_{T^*_-(e)}(a)\}$. Clearly, $a \notin T^*_-$. Then, $f_C(T_-) = f_C(T^*) + c[p_{T^*_-(e)}(a)] - c[(a)] < f_C(T^*) + c[p_{T^*}(a)] - c[(a)] = f_C[T^*_-(a)]$. This is a contradiction to the condition that $T^*_-(a) \in \mathcal{T}^*_-(a)$. Therefore, $c[p_{T^*_-(e)}(a)] \geq c[p_{T^*}(a)]$, and $u_{T^*_-(e)}(a) = f_C[T^*_-(a,e)] - f_C[T^*_-(e)] = c[p_{T^*_-(e)}(a)] - c[a] \geq c[p_{T^*}(a)] - c(a) = u_{T^*}(a)$.

In both cases, $u_{T^*_-(e)}(a) \geq u_{T^*}(a)$. □

The assumption that $T^*_-(e)$ is a maximum intersection tree is necessary, because there may exist $T^*_-(e) \in \mathcal{T}^*_-(e)$ and $a \in T^* \setminus \{e\}$, such that $a \notin T^*_-(e)$. By definition, the upper tolerance of such an edge a satisfies $u_T(a) = \infty$, whereas $u_{T^*}(a) = 0$.

Lemma 4.4.2 implies that, if $p_{T^*}(e_1) \neq p_{T^*}(e_2)$, then $u_{T^*_-(e_1)}(e_2) = u_{T^*}(e_2)$. This property is useful in tolerance-based BnB algorithms, because many upper tolerance values can be stored during branching steps instead of being recomputed. This storage of upper tolerance values may save large amounts of computing time.

We illustrate Lemma 4.4.2 with the following small example, depicted in Figure 4.7. Clearly, $T^* = \{(1,2),(2,3),(2,4)\}$ with $f_C(T^*) = 0$ is a unique MST. Moreover, $u_{T^*}(1,2) = 1$, $u_{T^*}(2,3) = 1$, $u_{T^*}(2,4) = 2$; $p_{T^*}(1,2) =$

$p_{T^*}(2,3) = (1,3)$, and $p_{T^*}(2,4) = (1,4)$. In the example, $(1,2)$ and $(2,3)$ have the same substitute edge $(1,3)$. When $(1,2)$ is forbidden, the substitute edge becomes $p_{T^*_-(1,2)}(2,3) = (3,4)$. As a consequence, the upper tolerance value of $(2,3)$ increases from 1 to 3.

Lemma 4.4.3. *Take any $e \in T^* \in \mathcal{T}^*$. Then for each $a \in T^* \setminus \{e\}$ such that $u_{T^*}(a) > 0$, it holds that $a \in \cap\mathcal{T}^*_-(e)$.*

Proof. If $\mathcal{T}_-(a,e) = \emptyset$, then $u_{T^*_-(e)}(a) = \infty$, meaning that $a \in \cap\mathcal{T}^*_-(e)$. So assume that $\mathcal{T}_-(a,e) \neq \emptyset$.

Assume to the contrary that there is a tree $T \in \mathcal{T}^*_-(e)$ such that $a \notin T$. Since $T \in \mathcal{T}^*_-(a)$ and $e \notin T$, it follows that $T \in \mathcal{T}_-(a,e)$.

Take any $T^*_-(a,e) \in \mathcal{T}^*_-(a,e)$ and any $T^*_-(a) \in \mathcal{T}^*_-(a)$. Lemma 4.4.2 implies that $u_{T^*_-(e)}(a) \geq u_{T^*}(a)$. Clearly, $f_C[T] \geq f_C[T^*_-(a,e)]$, since $T \in \mathcal{T}_-(a,e)$. So we have that $f_C[T] \geq f_C[T^*_-(a,e)] = f_C[T^*_-(e)] + u_{T^*_-(e)}(a) \geq f_C[T^*_-(e)] + u_{T^*}(a) > f_C[T^*_-(e)]$, which contradicts the condition that $T \in \mathcal{T}^*_-(e)$. □

Let $E = \{e_1, \ldots, e_k\} \subseteq T^*$. We denote and define the *upper tolerance of a set of edges* E as $u_{T^*}(E) = f_C[T^*_-(E)] - f_C[T^*]$. Given two sets $E_1, E_2 \subseteq T^*$, we have that, if $E_1 \subseteq E_2$, then $\mathcal{T}_-(E_2) \subseteq \mathcal{T}_-(E_1)$, therefore, $u_{T^*}(E_1) \leq u_{T^*}(E_2)$.

Lemma 4.4.4. *Let T^* be an MST and assume that $T^*_-(E) \neq \emptyset$ for some $E \subset \mathcal{E}$. Let $E = \{e_1, \ldots, e_k\}$. Then $u_{T^*}(E) = u_{T^*}(e_1) + u_{T^*_-(e_1)}(e_2) + \ldots + u_{T^*_-(e_1,\ldots,e_{k-1})}(e_k)$.*

Proof. Since $T^*_-(E) \neq \emptyset$, it follows for each $\bar{E} \subseteq E$ that $\mathcal{T}_-(\bar{E}) \neq \emptyset$. So all upper tolerance values $u_{T^*}(\bar{E})$ are finite.

Note that for each $j = 1, \ldots, k-1$, it holds that $u_{T^*_-(e_1,\ldots,e_j)}(e_{j+1}) = f_C[T^*_-(e_1, \ldots, e_{j+1})] - f_C[T^*_-(e_1, \ldots, e_j]$. Then,

$$\begin{aligned} u_{T^*_-(e_1,\ldots,e_k)} &= f_C[T^*_-(e_1,\ldots,e_k)] - f_C(T^*) \\ &= (f_C[T^*_-(e_1,\ldots,e_k)] - f_C[T^*_-(e_1,\ldots,e_{k-1})]) + \\ &(f_C[T^*_-(e_1,\ldots,e_{k-1})] - f_C[T^*_-(e_1,\ldots,e_{k-2})]) + \ldots \\ &\quad -f_C[T^*_-(e_1)] + (f_C[T^*_-(e_1)] - f_C(T^*)) \\ &= u_{T^*}(e_1) + u_{T^*_-(e_1)}(e_2) + \ldots + u_{T^*_-(e_1,\ldots,e_{k-1})}(e_k). \end{aligned} \tag{4.4.2}$$

□

Lemma 4.4.4 guarantees that for each subset $E \subset T^*$ and each $e \in T^* \setminus E$, $0 < u_{T^*_-(E)}(e) < \infty$, under the assumption that $\mathcal{T}_-(E \cup \{e\}) \neq \emptyset$.

Based on the four previous lemmas, we are able to construct new lower bounds for the STSP and the d-DCMSTP. Assume that the degree of a vertex $v \in V$ exceeds d. For instance, the maximum degree is 3, whereas the actual degree of v is 7. In order to 'reduce' the degree of v to 3, at least four edges incident to it have to be removed. According to Theorem 4.4.1, the minimum cost of removing four edges is at least equal to the sum of the smallest four upper tolerance values of edges incident to v.

More formally, let $v \in V$ be an arbitrary vertex in G and let $d_{T^*}(v)$ be the degree of v in T^*. Define $I_{T^*}(v) \subseteq T^*$ as the set of edges in the tree T^* incident to v. Order the edges in $I_{T^*}(v)$ in a non-decreasing order of upper tolerance values and obtain $u_{T^*}(e^1) \leq \ldots \leq u_{T^*}(e^{d_{T^*}(v)})$. Theorem 4.4.1 adds the lowest $d_{T^*}(v) - d$ upper tolerance values to $f_C[T^*]$ to obtain a lower bound of the d-DCMSTP solution value $f_C(D^*_d)$.

Theorem 4.4.1. *Let T^* be an optimal solution of the MSTP instance C, and let D^*_d be an optimal solution of the d-DCMSTP instance with the same C. Let $v \in V$ such that $d_{T^*}(v) > d$. Denote by $I^+_{T^*}(v) = \{e \in I_{T^*}(v) : \ u_{T^*}(e) > 0\}$ the set of edges incident to v with positive upper tolerance values. Let $k = |I^+_{T^*}(v)|$, and let $e^1, \ldots, e^k \in I^+_{T^*}(v)$ such that $0 < u_{T^*}(e^1) \leq \ldots \leq u_{T^*}(e^k)$. It then holds that $f_C(T^*) + \sum_{i=1}^{k-d} u_{T^*}(e^i) \leq f_C(D^*_d)$.*

Proof. Take any vertex v such that $d_{T^*}(v) > d$. By definition, $I_{T^*}(v) \cap D^*_d \leq d$, so at least $d_{T^*}(v) - d$ edges incident to v should be deleted in order to obtain a d-DCMSTP solution D^*_d. This implies that at least $k - d$ edges from $I^+_{T^*}(v)$ must be deleted.

There is a subset $E = \{e_1, \ldots, e_{k-d}\} \subset I^+_{T^*}(v)$ such that $E \cap D^*_d = \emptyset$. Clearly, $D^*_d \in \mathcal{T}_-(E)$, so that for each $T^*_-(E) \in \mathcal{T}^*_-(E)$, we have that $f_C[T^*_-(E)] \leq f_C[D^*_d]$.

Assume, without loss of generality, that $E \cap D^*_d = \emptyset$. It then holds that

$$f_C[T^*_-(E)] = f_C(T^*) + u_{T^*}(E) \leq f_C(D^*_d).$$

Since $D^d \in \mathcal{T}^*_-(E)$, it holds that $f_C[T^*_-(E)] \leq f_C(D^*_d)$. The term $u_{T^*}(E)$ can be bounded as follows:

$$u_{T^*}(E) = u_{T^*}(e_1, \ldots, e_{k-d}) =$$
$$u_{T^*}(e_1) + u_{T^*_-(e_1)}(e_2) + \ldots + u_{T^*_-(e_1,\ldots,e_{k-d-1})}(e_{k-d}) \quad (4.4.3)$$
$$\geq u_{T^*}(e_1) + \ldots + u_{T^*}(e_{k-d}) \quad (4.4.4)$$
$$\geq \sum_{i=1}^{k-d} u_{T^*}(e^i) \quad (4.4.5)$$

By Lemma 4.4.4, the value of $u_{T^*}(E)$ is independent of the order of deleting edges from E. It follows from Lemma 4.4.3 that $u_{T^*_-(e_1,\ldots,e_{i-1})}(e_i)$ is well defined for each $i = 2, \ldots, k-d$. Moreover, Lemma 4.4.2 states that $u_{T^*_-(e_1,\ldots,e_{i-1})}(e_i) \geq u_{T^*}(e_i)$ for each $i = 1, \ldots, k-d$. It follows from Lemma 4.4.2 that the term in (4.4.4) is at most equal to the term in 4.4.3. Finally, (4.4.5) holds, because we select the $k-d$ edges in $I^+_{T^*}(v)$ with the smallest upper tolerance values.

Since $u_{T^*}(E) \geq \sum_{i=1}^{k-d} u(e^i)$, it holds that $f_C(D^*_d) \geq f_C[T^*_-(E)] = f_C(T^*) + u_{T^*}(E) \geq f_C(T^*) + \sum_{i=1}^{k-d} u(e^i)$. □

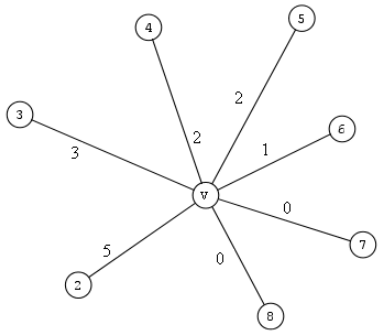

Figure 4.8. Vertex with seven incident edges in T^*

We illustrate Theorem 4.4.1 with the following example. In T^*, seven edges are adjacent to v, and the upper tolerance values are, in non-increasing order, $5, 3, 2, 2, 1, 0, 0$; see Figure 4.4. A DCMSTP solution with $d = 3$ is to be obtained. There are five edges in $I^+_{T^*}(v)$, so the sum of two smallest nonzero upper

tolerance values, $2+1$, is added to the value of the MSTP solution such as to form a lower bound for any 3-DCMSTP solution value $f_C(D_3^*)$.

In Theorem 4.4.1, we take the sum of the upper tolerances of $k-d$ edges $e \in I^+_{T^*}(v)$ instead of $d_{T^*}(v)-d$ edges from $I_{T^*}(v)$. The lower bounds obtained with both approaches are clearly the same. However, when only nonzero upper tolerances are considered, Theorem 4.4.1 is much easier to prove.

Theorem 4.4.1 can be applied in a similar way to construct a lower bound for both the STSP and the d-DCMSTP. The lower bound for the STSP is obtained by taking a vertex $v \in V$ with $d_{T^*}(v) > 2$, and by adding the $d_{T^*}(v) - 2$ smallest upper tolerance values of edges in $I_{T^*}(v)$. Note that only upper tolerances with respect to T^* need to be computed.

We conjecture that the lower bound can be improved further by not merely taking a single vertex into account, but each vertex with degree larger than d, i.e., all offenders. Let $T^* \in \mathcal{T}^*$, and $d \geq 2$. Let D_d^* be an optimal solution of the corresponding d-DCMSTP instance. Define $V^+ = \{v \in V : d_{T^*}(v) > d\}$. Denote the vertices in V^+ by $v_1, \ldots, v_{|V^+|}$. For each $v \in V^+$ and $S(v) \subseteq I_{T^*}(v)$, order the edges $e(S(v), i) \in S(v),\ i = 1, \ldots, |S(v)|$ in such a way that $u_{T^*}(e(S(v), 1)) \leq \ldots \leq u_{T^*}(e(S(v), |S(v)|))$. Define for each $v \in V^+$ the function $\Delta : T^* \to \mathbb{R}$ by

$$\Delta[S(v)] = \sum_{i=1}^{|S(v)|-d} u_{T^*}(e(S(v), i)) \tag{4.4.6}$$

We require $|S(v)| > d$, otherwise $\Delta[S(v)] = 0$. For each $v_k \in V^+$, let $W(v_k) := \{(v_i, v_k) \in I_{T^*}(v_k) :\ v_i \in V^+,\ i = 1, \ldots, k-1\}$.

Conjecture

$$f_C[T^*] + \sum_{v \in V^+} \Delta[I_{T^*}(v) \setminus W(v)] \leq f_C[D_d^*] \tag{4.4.7}$$

Clearly, when $|V^+| = 1$, the conjecture reduces to Theorem 4.4.1.

We illustrate the conjecture with the following example. Suppose that in T^*, $|V^+| = 2$, and $v, w \in V^+$. When $(v, w) \notin T^*$, then the conjecture states that both the $d_{T^*}(v) - d$ smallest upper tolerances incident to v and the $d_{T^*}(w) - d$ smallest upper tolerances of edges incident to w can be added to the lower bound.

A problem arises when for $v, w \in V^+$, the edge e between v and w belongs to T^*, i.e., $e = (v, w) \in T^*$. In order to prevent double-counts, (v, w) should be assigned to either the incidence set of v or that of w. Take $d = 3$, $d_{T^*}(v) = 4$, $d_{T^*}(w) = 6$, and $(v.w) \in T^*$. Suppose that the upper tolerance values of the edges incident to v are $5, 3, 3, 1$, of the edges incident to w $4, 4, 3, 2, 2, 0$, and suppose that $u_{T^*}(e) = 3$. If we assign e to v, then $W(w) = \{e\}$ and $W(v) = \emptyset$. It means that sum of the smallest upper tolerance value of the edges in $I_{T^*}(v)$ and the two $(= d_{T_*}(w) - d - 1)$ smallest upper tolerance values from edges in $I_{T^*}(w) \setminus \{e\}$, can be added to the value of $f_C[T^*]$. The value of the lower bound is then $f_C[T^*] + 1 + (2 + 0) = f_C[T^*] + 3$. If, however, we assign e to $W(w)$ ($W(w) = \emptyset$), then the lower bound is $f_C[T^*] + 0 + 2 + 2 = f_C[T^*] + 4$. The pseudo-code for constructing the conjectured lower bounds is given in Algorithm 1.

Algorithm 1 Method for constructing a lower bound for the d-DCMSTP

INPUT
  d User-specified maximum degree of each vertex;
  C Cost matrix;

MAIN ALGORITHM
  Compute MSTP solution T^*;
  Obtain for each $v \in V$ the value of $d_{T^*}(v)$; obtain $f_C(T^*)$.
  lb := $f_C(T^*)$.
  Let V^+ be the set of vertices s.t. $d_{T^*}(v) > d$;
  Make an ordering $v_1, \ldots, v_{|V^+|}$ of the vertices in V^+.
  for $i = 1$ **to** $|V^+|$
    for each $j > i$ s.t. $(v_i, v_j) \in T^*$
      assign (v_i, v_j) to v_i;
      $d_{T^*}(v_j) := d_{T^*}(v_j) - 1$;.
      $I_{T^*}(v_j) := I_{T^*}(v_j) \setminus \{(v_i, v_j)\}$; (Comment: subtract $W(v_j)$ from $I_{T^*}(v_j)$)
    end;
    for each $e \in I_{T^*}(v_i)$
      compute $u_{T^*}(e)$;
    Order upper tolerance values of edges $e \in I_{T^*}(v_i)$ in non-increasing order;
    $sumtol$:= sum of $d_{T^*}(v_i) - d$ smallest upper tolerance values;
    lb := $lb + sumtol$;
  end;
end.

OUTPUT

Lower bound lb.

Algorithm 1 depends on the order in which the vertices are considered. In the example above, the lower bound is $f_C[T^*] + 4$ when $e \in W(w)$ and $f_C[T^*] + 3$ when $e \in W(v)$.

Table 4.2. Quality of new bound for the STSP

Instance	MST	1-Tree	New bound	STSP	Gap 1-Tree STSP bridged
att48	2698	2737	7344	10628	58.38%
bays29	1557	1623	1696	2020	18.39%
berlin52	6066	6319	6432	7542	9.24%
bier127	94680	94939	95987	118282	4.49%
brazil58	9648	10128	10579	25395	2.95%
eil101	528	536	546	629	10.75%
eil51	359	369	378	426	15.79%
eil76	454	463	471	538	10.67%
fri26	741	834	841	937	6.80%
kroB100	19203	19332	19631	22141	10.64%
kroC100	18354	18642	18932	20749	13.76%
kroD100	18552	18756	19138	21294	15.05%

We have chosen a lexicographic strategy in the experiments below, i.e., the first vertex in the matrix is ranked first. For future research, other strategies should be considered, such as an ordering of the vertices in non-decreasing order of their degrees in T^*.

Table 4.2 shows the results of our conjectured bound for symmetric instances from the TSPLIB (Reinelt, 1991). The *1-Tree* is obtained as follows. One vertex, say v^*, is removed from the vertex set V and the edges incident to v^* are removed from $\mathcal{E}$. The MSTP problem is solved on the new instance with $n-1$ vertices. Finally, the two lowest cost edges incident to v^* are added to the MSTP solution and the 1-Tree is obtained. The "new bound" in Table 4.2 first constructs a 1-tree and then constructs the lower bound described in the conjecture. Except for vertex v^*, the procedure is the same as the one described in Algorithm 1. The last column reports the percentage of the gap between the values of the 1-Tree bound and the value of the optimal STSP solution bridged by the conjectured lower bound.

The results on these small instances show that only a small part of the gap between the MST solution and the STSP solution is bridged with the upper bound. For the STSP, the usual bound from Held and Karp (1970) is very tight. For the DCMSTP, however, the MST lower bound is often still used, although in Volgenant (1989), a Lagrangian relaxation is used to tighten the lower bound. When

it is proven that the conjecture is true, the application of the lower bound to the DCMSTP is a promising direction of research.

4.5 Conclusion

In this chapter, we discuss additional topics on tolerance-based algorithms.

In Section 4.2, we have conducted experiments on randomly generated ATSP instances. It is found in, among others, Zhang (1993), that there is a thin boundary between $\mathcal{NP}$-hard problem instances that are difficult by nature and instances that are easily solvable. The difficulty depends on the number of different intercity distances, which in turn depend on the range R of the uniform distribution from which the distances are drawn. The experiments indicate that an increase in R leads to far smaller increases in solution times of our tolerance-based BnB algorithm than our cost-based benchmarks.

Section 4.3 determines at which nodes in the BnB search tree the use of tolerances effective. Section 3.6 shows that predictions based on tolerances are more accurate than predictions based on costs. In Depth First Search algorithms the choice at top nodes of the search tree is crucial. For this reason, we restrict the use of tolerances to top nodes of the tree, and apply cost-based branching and bounding to other nodes. Our computational experiments indicate that for a minority of instances these hybrid BnB algorithms are more effective than the pure tolerance-based variants. However, more experiments are needed to gain better understanding of this phenomenon.

Finally, in Section 4.4, we discuss procedures for increasing the tolerance-based lower bounds from Section 3.6. Instead of adding the upper tolerance value of a single element, we propose a procedure for adding the sum of several elements to the value of the relaxation solution. Such a procedure is generally invalid for the ATSP and its AP relaxation. However, we show that the procedure can be applied successfully for Minimum Spanning Tree Problems, which are relaxations of Symmetric TSPs and Degree-Constrained Minimum Spanning Tree Problems.

Appendix

Table 4.3. Search tree sizes for various values of δ

	Level							
Instance	0	2	5	10	20	100	1000	∞
ft53	20111	15967	20671	27403	95173	95045	30831	19200
ftv33	7065	2570	3741	3570	3378	1732	1362	1362
ftv38	6195	11894	12005	11976	11602	8080	2091	2091
ftv44	619	824	793	764	368	171	171	171
ftv47	29025	29407	33342	31739	33817	31605	7692	7692
ftv55	92447	162421	185662	180543	180339	171781	108583	23034
ftv64	43441	43788	143165	145612	145202	132776	60546	15509
ftv70	253873	165683	202568	169157	167525	156989	46021	17694

Table 4.4. Solution times for various values of δ

	Level							
Instance	0	2	5	10	20	100	1000	∞
ft53	2.31	1.70	2.20	2.86	10.16	10.49	6.59	6.70
ftv33	0.22	0.05	0.11	0.11	0.11	0.11	0.11	0.16
ftv38	0.22	0.27	0.22	0.27	0.27	0.38	0.38	0.38
ftv44	0.00	0.05	0.05	0.00	0.05	0.05	0.05	0.00
ftv47	1.26	1.15	1.32	1.32	1.43	1.43	1.54	1.54
ftv55	4.51	6.70	7.64	7.47	7.47	7.53	7.75	9.51
ftv64	3.19	2.64	7.47	7.64	7.58	7.91	10.00	11.10
ftv70	24.95	13.96	15.60	13.13	13.13	13.02	9.84	8.68

Chapter 5

Balancing the Fit and Logistics Costs of Market Segments

5.1 Introduction

The key to success in the customer-oriented philosophy is the ability to understand customer needs (Kotler and Armstrong, 2006). These needs usually vary widely between consumers, a phenomenon which is called *consumer heterogeneity* (Allenby *et al.*, 1998). Since it is in general too expensive to serve each consumer separately, companies identify groups of customers, *segments*, with approximately the same relevant characteristics (Wedel and Kamakura, 2002). The process of designing segments is called *segmentation*; its purpose is to construct groups of consumers that are similar within groups and dissimilar between groups. Companies select their target segments and develop strategies to fulfill the needs of the customers in these segments (Smith, 1956), so that advantages over competitors are obtained. The requirement is that the marketing mixes, that the company deploys, fit individual consumer's preferences as much as possible.

Most segmentation studies maximize the fit of the segmentation. However, when consumers in the same segment are located far apart, the logistics costs for the organization can become very high. Steenkamp and Ter Hofstede (2002) report that "geographically dispersed consumer segments will often not allow profitable entry strategies to be pursued." This is in particular true "in industries where distribution costs constitute a large part of the total costs, such as in retailing and in industries dealing with perishable products" (Steenkamp and

Ter Hofstede, 2002). The importance of logistics costs is growing as a result of the trend of globalization (Steenkamp and Ter Hofstede, 2002). Although studies observe that transnational segments exist (Yavas *et al.*, 1992; Ter Hofstede *et al.*, 1999), many companies still target nations or groups of nations as separate segments; see for example Salah and Pervel Kathanis (1994).

When logistics costs are kept low, the resulting segmentations may fit consumer preferences worse than a standard segmentation strategy. However, this effect need not be large. In the segmentation study Ter Hofstede *et al.* (2002), it is found that, geographically close segments still represent consumer heterogeneity well.

These arguments form a motivation for constructing segments which may be less similar in their characteristics, but which are more similar in geographical location. We balance the maximization of the consumer benefits with the minimization of the logistics costs of supplying the products.

In most modern day segmentation studies, large populations of consumers are considered. Segments are identified, and an appropriate marketing mix is designed for each of them; see for example Helsen *et al.* (1993); Ter Hofstede *et al.* (1999); Bijmolt *et al.* (2004). The state-of-the-art tool for solving these problems is *mixture modeling* (Wedel and Kamakura, 1998). Mixture models use a separate statistical distribution for each segment, with which the preferences of individual consumers are modeled. Since the properties of individual consumers are represented in mixture models, consumer heterogeneity is well preserved. Similar types of segmentation approaches are *Bayesian modeling* (**?**) and *mixed models* (**?**). The results are not just averaged, but each individual observation in a segment is preserved in the statistical distribution. This conceptual advantage has recently given statistical methods, such as mixture modeling, the upper hand over other methods (Ter Hofstede *et al.*, 1999; Wedel and Kamakura, 1998).

We consider in this paper the special class of segmentations as described in Boone and Roehm (2002), where the company not only constructs marketing mixes for each chosen segment, but also assigns each individual consumer to a segment. Examples of this type of segmentation are also provided in Boone and Roehm (2002). The performance of mixture models depends strongly on the data. According to Wedel and Kamakura (1998), “mixture models are sensitive to poorly defined and separated segments, large numbers of local maxima, and

other violations of its underlying assumptions". Boone and Roehm (2002) find that the performance of mixture models is poor: many consumers are assigned incorrectly. This incorrect assignment leads to a reduction in the effectiveness of a segmentation strategy and squanders marketing resources. In this case, *hard non-overlapping* cluster methods are the most appropriate ones.

Hard cluster methods assign subjects deterministically to segments; non-overlapping methods assign each subject to exactly one segment (Punj and Stewart, 1983). Commonly used non-overlapping methods are *Ward's* algorithm (Ward, 1963) and the k*-means* algorithm (MacQueen, 1967). Cluster algorithms are usually heuristics; exact algorithms are hardly applicable to clustering problems, since they require so much solution time that only small instances can be solved (Du Merle *et al.*, 2000). Recently, there has been a major development in the application of *meta-heuristics* to segmentation problems, such as simulated annealing (Klein and Dubes, 1989; Brusco *et al.*, 2002), and artificial neural networks (Boone and Roehm, 2002). For an overview of hard non-overlapping clustering methods, we refer to Jain *et al.* (1999) and Mirkin and Muchnik (1998). The *NORMCLUS* cluster methods, presented in DeSarbo and Grisaffe (1998), enable the user to incorporate a broad set of possible constraints on segments, such as minimum size of segments and maximum acceptable difference between two subjects within a segment.

5.2 The logistics costs of segmentations

The segmentation decision and the decision on the design of the supply chain are usually taken independently, and until now, segmentation modeling and logistics modeling have been two disconnected fields of research. From Steenkamp and Ter Hofstede (2002), it can be concluded that large distances form the main cause of high logistics costs of a segmentation. This is a motivation for considering the logistics costs of segmentations.

When distances between consumers in the same segment are large, the logistics costs are high, in particular for perishable goods; see Steenkamp and Ter Hofstede (2002). Longer distances lead to longer lead times, more inventory, and more variability in the orders; see Nelson and Toledano (1979). The following figures illustrate the importance of logistics costs. In Davis (1990), it is found

that, on average, the cost of the physical distribution makes up about 22% of the total cost of a product. Recently, the GMA (Grocery Manufacturers Association) has found that transportation costs accounts for 62% of the logistics costs in the American grocery sector, and that transportation costs have been rising steadily (GMA, 2005). In this section, we identify the logistics costs associated with the choice for a segmentation.

We assume that for each segment a separate facility is established. This assumption appears to be realistic for our case study on retailing; the European Council on Applied Sciences and Engineering (Euro-CASE) has conducted research on supply chains in European retailing (Euro-CASE, 2001). The report observes that stores used to be supplied from a retailer-controlled Regional Distribution Centers (RDCs). Currently, there is a trend towards European Distribution Centers (EDCs), which are opened to establish Pan-European supply networks. This means that one central EDC supplies stores all over Europe, That is, if it is not too costly to transport the commodities over large distances. For some products, the logistics costs of usual transnational segmentations are too high. If Pan-European EDCs are established, the distances from those Pan-European facilities to individual stores are too large, leading to high transportation costs. RDCs are then usually maintained.

We introduce a measure for quantifying the logistics costs, namely the *Distance to a Central Facility (DCF) measure*. The logistics costs are estimated with the sum of the distances to a central facility in each segment. When the DCF measure is used, the underlying assumption is that a depot for each segment is built in a central position, and that each subject is served from this central facility. Consumers within each segment are then continuously replenished from the central location. The DCF measure is rooted in the *center of gravity* model, introduced in Francis and White (1974). The model locates all demand points on a two-dimensional plane; the weight of a demand point is the expected demand in that point. The virtual facility is then located in the center of gravity of all demand points. Note that the purpose of our model is not to establish optimal locations for facilities, but to estimate the logistics costs. In our measure, the logistics costs are proportional to the sum of the distances from all consumers to the location of the central facility. This is graphically depicted in Figure 5.1.

The following small example illustrates the DCF measure. Assume that there

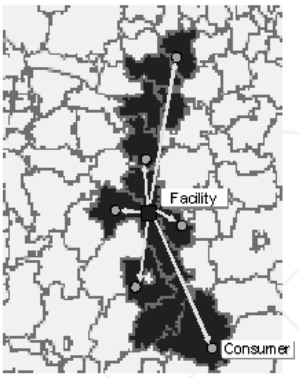

Figure 5.1. Example of DCF measure of logistics costs

are three consumers in a segment, all with equal demands. Assume that the consumers are located in a two-dimensional plane on the coordinates $(0, 0)$, $(2, 0)$ and $(1, 3)$, respectively. The central position is $(1, 1)$, so the the DCF measure is the sum of the distance between the consumer demand points and the central facility, being $\sqrt{2} + \sqrt{2} + 2 \approx 4.828$.

The following extensions can be made to the logistics costs model; see Krarup and Pruzan (1989).

- We assume that one central facility supplies all consumers in the corresponding segment. If more than one facility is needed, then we obtain an capacitated p-median problem, where p denotes the number of facilities; Mirchandani and Francis (1989).

- The DCF logistics costs measure places the facility on a central location. Other considerations for the location of the facility, listed in Kotler *et al.* (1993), are not taken into account.

- The DCF measure only consists of the transportation costs. The costs of fixed facilities, such as warehouses, are not included, because they are more or less independent of the chosen segmentation.

- We assume in our case study from Section 5.5 that the demand of consumers are more or less equal. If, however, reliable demand estimations are known, the facility is shifted towards consumers with larger demands.

An example of a reliable demand estimation is the All Commodity Value (ACV): the value of all commodities bought in a region.

5.3 The Budget Constraint Approach

We have seen that, due to high logistics costs, many companies resort to a segmentation strategy which takes countries or regions as segments (Salah and Pervel Kathanis, 1994; Steenkamp and Ter Hofstede, 2002). Such a traditional segmentation ignores the similarity of consumers across boundaries and the dissimilarity of consumers within boundaries found in, among others, Ter Hofstede *et al.* (2002) and Yavas *et al.* (1992). Is it possible to serve transnational segments well against reasonable logistics costs?

Ideally, the segmentation strategy with the highest expected profit level is selected, where profit is defined as the expected revenues of the segmentation minus the costs. However, direct profit maximization is not possible, because it is not known how much the sales increase when the fit of consumer preferences increases. Although it is well known that an increase in the fit of consumer preferences has a positive effect on sales (Simonson, 2005; Kumar and Petersen, 2005), none of the studies actually quantify this relationship.

We therefore choose to restrict the logistics costs. Within a given *budget* B, we try to find the best segmentation with the DCF logistics costs at most B. An alternative is the restriction on the maximum distance between two subjects in the same segment; see Section 5.8.

DeSarbo and Grisaffe (1998) present the NORMCLUS framework for solving clustering problems, with which several requirements on segments can be imposed. For example, to prevent an organization from serving small segments at a loss, a minimum can be imposed on the segment size, and the maximum distance of each consumer in a segment to a fixed location can be restricted. The objective function is flexible, so that it can accommodate the simultaneous optimization of multiple criteria. However, when the NORMCLUS framework is used to simultaneously maximize the fit and minimize the logistics costs of a segmentation, two problems arise. Firstly, the logistics costs constraint is imposed on the entire segmentation. This is a new and very specific type of constraint, requiring geographical information of each customer and the computation of

central locations of every solution considered in the search process. The second problem is that the weight of the logistics costs compared to the fit of the segmentation is hard to determine in advance. It is difficult, or even impossible, to know in advance which logistics costs budget leads to a segmentation with the highest profit. It is then better to vary the budget and compute a number of alternative segmentations.

To make the trade-off between lower logistics costs and an increase in the fit of consumer preferences, we introduce the *Budget Constraint Approach*. The approach constructs segmentations for different values of B, resulting in a large set of candidate segmentations. From this set, we then choose a few viable alternatives. The process is summarized in Figure 5.2.

The Budget Constraint Approach comprises two major steps:

Step 1: Construction of candidate segmentations;

Step 2: Reduction of the number of alternatives.

The set of candidate segmentations is formed as follows. Two input parameters jointly influence the quality of the solution values, namely the number of clusters K and the budget B. The Budget Constraint Approach solves a sequence of clustering problems for which the value of B is gradually increased. The solution obtained for a given value of B is used as starting solution for the next clustering problem in the sequence. The use of starting solutions from previous steps is only possible when the budget is gradually increased: a solution satisfying a budget B automatically satisfies every budget $B^* > B$. As a consequence of this procedure, no new starting solutions need to be computed. It is important to make an appropriate choice of the step size between two subsequent budgets. If the increase in B is too large, we take the risk that promising segmentation solutions are overlooked; if the increase in B is too gradual, the procedure becomes very time consuming. If the number of clusters is high, it is easier to construct clusters with relatively small B values and at the same time achieve a high level of homogeneity within the segments; see Section 5.4. Therefore, the clustering problem is not only solved for the number of segments obtained by a standard clustering method, but also for slightly bigger values of K.

The following clustering problem is solved for each selected combination of K and B. Define $x_{ik} = 1$ if subject i belongs to cluster k, and $x_{ik} = 0$ otherwise.

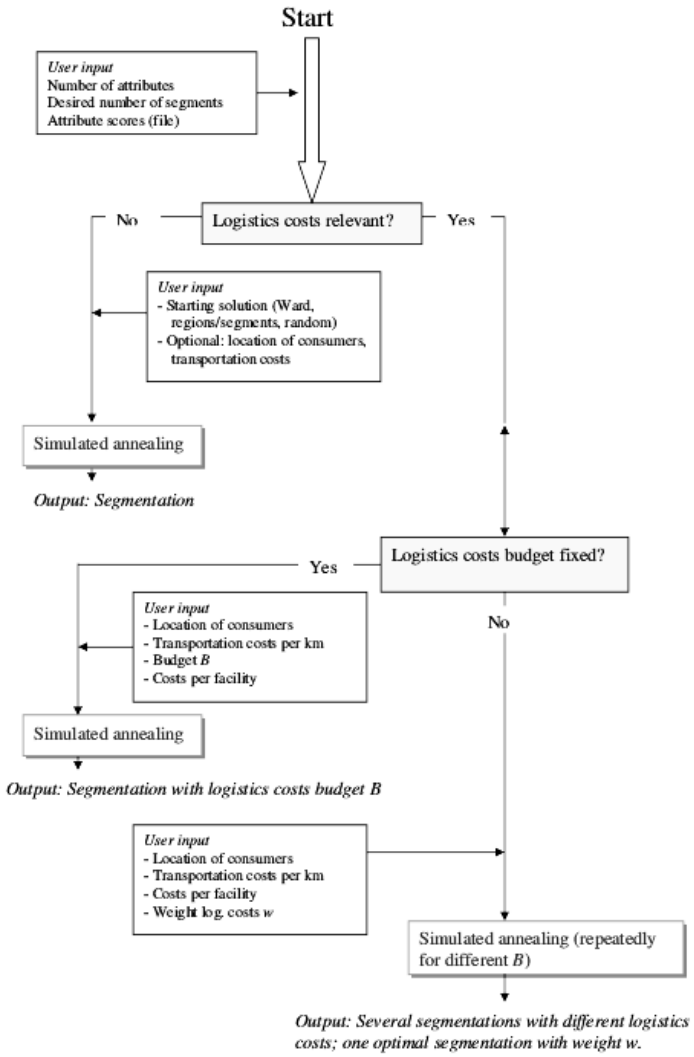

Figure 5.2. Flowchart of the Budget Constraint Approach

$$\begin{aligned}
\min \quad & \textstyle\sum_{i=1}^{N}\sum_{k=1}^{K} x_{ik}\|f_i - z_k\|^2 && && (5.3.1)\\
s.t. \quad & \textstyle\sum_{k=1}^{K} x_{ik} = 1 && i = 1,\ldots,N && (5.3.2)\\
& \textstyle\sum_{k=1}^{K}\sum_{i\in C_k} d_{ig_k} x_{ik} \le B && && \\
& x_{ik} \in \{0,1\} && i = 1,\ldots,N && \\
& && k = 1,\ldots,K && (5.3.3)
\end{aligned}$$

Here, f_i is the vector of attribute scores of subject i, and z_k is the vector of attribute scores of an average subject of cluster k. Moreover, d_{ij} denotes the geographical distance between two subjects i and j, and d_{ig_k} denotes the distance from subject i to the center of gravity of segment k. C_k denotes the set of subjects i that belong to cluster k in a given solution. The objective function, (5.3.1), is the commonly used *Minimum Sum-of-Squares Criterion (MSSC)* (Du Merle *et al.*, 2000; Milligan and Cooper, 1987), which minimizes the sum of the squared distances to an average subject of each segment. Constraint (5.3.2) requires that each consumer is assigned to exactly one segment, and constraint (5.3.3) requires that the total logistics costs do not exceed B.

Due to the constraints on the clusters, the problems belong to the class of so-called *constrained clustering problems*; see Everitt *et al.* (2001); DeSarbo and Grisaffe (1998). These problems may be more difficult to solve than unconstrained clustering problems, firstly, because it is hard to find a good starting solution. For example, Ward's algorithm is often not able to construct a starting solution at all, even when B is relatively large. Secondly, many moves become unavailable for small values of B, because those moves lead to clusters with logistics costs larger than B. If only improving moves are allowed, such as in k-means, then it is very likely that bad local optima are found for low values of B.

We solve the Budget Constraint clustering problems with *simulated annealing*. Wedel and Kamakura (1998, p.54) report that simulated annealing is an "especially appealing approach to overcome the local optimum problem". Annealing is the process in physics occurring when a substance is heated such that particles arrange themselves randomly. The temperature is lowered slowly, so that the particles stabilize and the desired structure emerges; see Van Laarhoven

and Aarts (1987). Simulated annealing in physics is the simulation of this annealing process. It evaluates a sequence of states; the step from one state to another is called a *move*. Kirkpatrick *et al.* (1983) describe the analogy between the annealing of a substance and optimization: initially the temperature of the cooling schedule is high, and many random perturbations of the solution at hand are allowed, even if they lead to moves towards worse solutions than the one at hand. When a promising subset of solutions is established, the temperature is lowered, meaning that a smaller number of perturbations is allowed. Finally, the freezing temperature is reached and a good or optimal solution is returned. Simulated annealing is applied to solve clustering problems in Klein and Dubes (1989) and Brusco *et al.* (2002), and also for many other problems; see Henderson *et al.* (2003) and the references therein.

The simulated annealing algorithm for solving clustering problems is described below. The algorithm proceeds through a sequence of cluster solutions. It randomly changes the cluster membership of one subject in the current solution. If the new cluster solution has a lower MSSC score, then the algorithm proceeds to this solution. Otherwise, if the MSSC score of the new solution is worse than the score of the current solution, the move to the new solution is made with a probability that depends on the magnitude of the deterioration and on a parameter value of the algorithm, called the *temperature*. Typically, the temperature is initially high, so that many deteriorating moves may be made. Later on in the process, the temperature is decreased. The simulated annealing algorithm described below is similar to the algorithm described in Brusco *et al.* (2002).

The algorithm makes moves from a current solution to solutions in the neighborhood in which the cluster membership of a single subject is altered. However, only some of these solutions in the neighborhood of a current solution satisfy the budget restriction. The algorithm computes the logistics costs of a new solution S and compares it to the user-defined budget B. The algorithm can also accommodate other requirements on segments, such as a minimum segment size.

After having obtained a set of cluster solutions for various values of K and B, we enter the second step of the approach: the reduction of the number of alternatives. We try to find solutions which combine a good score on the fit of the data with a small number of clusters and low logistics costs. The relative importance of each of these factors determines which segmentation is the best

Algorithm 2 Simulated annealing approach for budget-constrained clustering

INPUT

S	Segmentation;
$c(S)$	MSSC cost of a segmentation S;
$l(S)$	logistics costs of a segmentation S;
B	logistics costs budget;
K	number of segments.

PARAMETERS

T_0	initial temperature;
α	cooling parameter;
$STABLE$	number of tries before stability at temperature T is achieved;
T_f	temperature at which the algorithm is frozen.

MAIN ALGORITHM

Generate initial cluster solution S^* such that $l(S^*) \leq B$;
$T := T_0$;
repeat
  count := 0;
  repeat
    randomly select cluster solution $S := findMove(S^*)$;
    $\delta = c(S^*) - c(S)$;
    if $\delta \geq 0$
      $S^* := S$;
    else
      rnd := random number from $U(0, 1)$;
      if $rnd < \exp\{\frac{\delta}{T}\}$
        $S^* := S$;
    count := count + 1;
  until count = STABLE;
  $T := \alpha \times T$;
until $T < T_f$;

FUNCTION findMove(S^*)
feasible := **false**;
while not feasible
  S := perturb S^* by randomly changing the membership of a randomly chosen subject n;
  if $l(S) \leq B$ feasible := **true**;
return S.

OUTPUT

(Almost) optimal cluster solution S^* with cost value $c(S^*)$.

one for the organization. Unfortunately, it is very difficult to translate the fit of the clustering solution into the corresponding expected profit when a segmentation is based on preferences and perceptions. This is true for the case study from Section 5.5. As a consequence, we cannot attach fixed weights to the objectives and choose an 'optimal' segmentation.

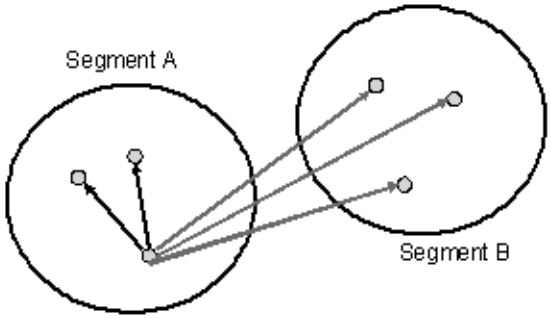

Figure 5.3. The silhouette width of a subject

We evaluate the alternatives on two criteria: the silhouette width measure of fit, and the logistics costs. The *silhouette width* of subject i divides the average distance to other subjects in the segment to which i belongs to the average distance of subject i to subjects in other segments; see Rousseeuw (1987). The distance in this context is between the attribute scores of two subjects, and not the geographical distance. The silhouette width ranges between -1, the worst possible score, and +1, the best possible score.

The silhouette width is computed as follows. Assume that subject i belongs to segment A. Then, let $a(i)$ be the average distance of i to all other subjects in A. Moreover, let $d(i, B)$ denote the average distance of i to all subjects in cluster B, and let $b(i) = \min_{B \neq A} d(i, B)$. Then the silhouette width $s(i)$ of subject i is:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{5.3.4}$$

The *silhouette width of a segmentation* is the average silhouette width over all consumers. Other measures, such as the value of the maximum likelihood of mixture models and the MSSC score, are either non-increasing or non-decreasing with the number of clusters, i.e., the larger the values of K, the better the score

of the cluster solution. The value of the silhouette width, however, does not necessarily improve with increasing values of K. Rousseeuw (1987) claims that the silhouette width attains its highest value for the number of clusters that describes the data in the best way, i.e., the largest similarity within segments and the smallest dissimilarity between segments. Some refined measures of the classification likelihood, such as the so-called AIC3, BIC, and CAIC, share this favorable property; see for example **?**.

An important decision in segmentation studies is the choice of the number of segments. An increase in the number of segments generally leads to better fitting segments, but this is at the expense of higher costs: new marketing strategies must be developed and segment facilities need to be set up. Methods for determining the number of segments are described in Milligan and Cooper (1987). A usual method is *visual inspection*: the number of segments is chosen in such a way that increasing the number of segments does not improve the the quality of the cluster solution substantially. In our approach, the best number of segments needs to be determined for each considered logistics costs budget. Instead of visual inspection, we choose the value of K in such a way that the resulting segmentation achieves the highest silhouette width. We assume that if the silhouette width increases when a new segment is added, the profit gained, as a result of the increase in the fit of consumer preferences, is sufficient to outweigh the additional costs of new segments.

After determining the best number of segments for each logistics costs budget, we vary the relative weight w of the logistics costs compared to the silhouette width. The final score of an alternative S is denoted by $F(S; w)$ and defined as:

$$F(S; w) := (1 - silhouette\ width) + w \times logistics\ costs. \tag{5.3.5}$$

The segmentation S^* is called *efficient* if, for some w, the value of $F(S^*; w)$ is lower than the values of $F(S; w)$ for all other considered segmentations $S \neq S^*$. The Budget Constraint Approach returns the set of efficient segmentations. The user can then specify his or her own importance of the logistics costs, and make the trade-off between an acceptable logistics costs level and good fit.

Table 5.1. Random instances by Milligan (1985)

Name	Type of clustering problem
TYPE 1	Error free data
TYPE 2	20% outliers
TYPE 3	40% outliers
TYPE 4	Error perturbed coordinates (low level)
TYPE 5	Error perturbed coordinates (high level)
TYPE 6	Added random noise dimension
TYPE 7	Two added random noise dimensions
TYPE 9	Standardized coordinates
TYPE 11	Random noise data without cluster structure

5.4 Experiments with randomly generated cluster instances

In this section, we evaluate the simulated annealing approach of Section 5.3 on randomly generated cluster instances. We also examine for which types of clustering problems the Budget Constraint Approach is able to obtain well-fitting segmentations with moderate logistics costs.

We use the randomly generated artificial instances from Milligan (1985). The attribute scores are drawn from normal distributions, but many instances have nasty properties, or 'errors'. The included errors are outliers, additional noise variables, unequal cluster sizes and erroneous observations; see Table 5.1. These properties, which are also encountered in practical situations, make cluster algorithms less accurate. The instances from Milligan (1985) have been used in comparative studies of algorithms, such as Boone and Roehm (2002) and Milligan and Cooper (1987).

The true clustering solutions of the instances of Milligan (1985) are known, so that the solutions obtained by any algorithm can be compared to the true cluster solutions. The Adjusted Rand Index (ARI), introduced in Hubert and Arabie (1985), measures how similar a given cluster solution is compared to the 'true' cluster solution. More precisely, it determines the fraction of pairs of subjects which are in the same cluster in both the 'true' and the given clustering. A value of 1 indicates that the algorithm has returned the true cluster solution; a value of 0 means that the cluster solutions at hand has the quality of an average random classification. Milligan and Cooper (1986) report that the Adjusted Rand Index is the most accurate criterion for measuring solution quality.

The Adjusted Rand Index (ARI) is defined as follows. Suppose there is a set of objects O and there are two partitions $L = L_1, \ldots, L_R$ and $M =$

$M_1, \ldots, M_C$ of O such that $\bigcap L_i \cap L_j = M_i \cap M_j = \emptyset$ for each pair of subsets $L_i \neq L_j$ of L and $M_i \neq M_j$ of M. So $O = L_1 \cup \ldots \cup L_R = M_1 \cup \ldots \cup M_R$. Assume that $|O| = n$. Construct the following $R \times C$ matrix, in which element n_{ij} denotes the number of elements which are both in class L_i and in class M_j. Furthermore, let $n_{i.}$ and $n_{.j}$ denote the row and column totals of this matrix, i.e., the number of elements in class L_i and M_j, respectively. Then:

$$ARI = \frac{\sum_{i,j} \binom{n_{ij}}{2} - [\sum_i \binom{n_{i.}}{2} \sum_j \binom{n_{.j}}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}] - [\sum_i \binom{n_{i.}}{2} \sum_j \binom{n_{.j}}{2}]/\binom{n}{2}} \tag{5.4.1}$$

In Boone and Roehm (2002), experiments are performed on the instance types 1, 2, 4, 6, and 7 from Table 5.1 with k-means, artificial neural networks and mixture models. The k-means method (with Ward's algorithm to generate initial solutions) and artificial neural networks perform approximately equally well. Mixture models achieve lower average scores on all tested types of instances. Since k-means turns out to be an effective cluster method in Boone and Roehm (2002), we compare simulated annealing to this method.

Table 5.2. Average Adjusted Rand Index of simulated annealing and k-means

	SA Ward	k-means Ward	SA Random	k-means Random
Type 1	0.999	0.999	0.936	0.512
Type 2	0.714	0.618	0.698	0.399
Type 3	0.530	0.519	0.521	0.434
Type 4	0.988	0.987	0.920	0.487
Type 5	0.896	0.861	0.845	0.418
Type 6	0.711	0.694	0.620	0.400
Type 9	0.986	0.986	0.936	0.791

In Table 5.2, the adjusted Rand indices of simulated annealing (SA) and k-means are shown, both with randomly generated starting solutions and with starting solutions obtained with Ward's algorithm. When Ward's algorithm is used to generate starting solutions, simulated annealing achieves only marginally better results than k-means for all types of instances. The role of Ward's algorithm is very important here: it constructs good starting solutions and even retrieves the true clustering solutions for many test instances. Our simulated annealing algorithm requires longer solution times to obtain the improvements: 0.1 to 4.5

Table 5.3. Results for random cluster instances with increasing budget

		Logistics costs level			
Instance	Comment	Measure	Low	Intermediate	High
Type 1	Normal,	ARI	0.033	0.636	0.937
	random	Log. costs	2211	3467	3643
Type 3	40% outliers	ARI	0.004	0.189	0.536
		Log. costs	4310	5736	6830
Type 4	Coordinates	ARI	0.033	0.626	0.941
	perturbed	Log. costs	2210	3457	3625
Type 5	Coordinates	ARI	0.032	0.559	0.848
	perturbed	Log. costs	2207	3446	3611
Type 6	Added noise	ARI	0.023	0.449	0.628
	dimension	Log. costs	2218	3454	3611
Type 9	Standardized	ARI	0.028	0.900	0.954
	data	Log. costs	2268	3640	4567

seconds, where the average is 2.225 seconds and the median 2.418 seconds. On the other hand, the solution times of k-means are usually within 0.1 second. However, when the starting solutions are randomly generated, simulated annealing achieves clearly better solutions than k-means. These findings confirm the conclusion in Wedel and Kamakura (1998) that simulated annealing is less dependent on the starting solution. We also find that, if the value of B is small, Ward's algorithm is often not able to construct a feasible solution.

To include the logistics costs in the experiments, we disperse the locations of the subjects in a 100 by 100 plane and assume that the logistics costs are proportional to the distances in this plane. An initial solution is obtained by grouping closely located subjects into segments. The logistics costs budget is then gradually increased until an unconstrained segmentation is obtained.

The Budget Constraint Approach is particularly useful when good segmentations with relatively low logistics costs exist. Is this true for the randomly generated instances? Of each type reported in Table 5.3 and for each logistics costs level, 48 cluster instances are solved. The fit decreases when the logistics costs budget is cut down. However, for the standardized random data of type 9, the solution with intermediate logistics costs is almost as good as the unconstrained solution, but with much lower logistics costs.

In the experiments on random instances, we use the Adjusted Rand Index to compare the classifications obtained by the tested algorithms with a 'true'

classification. Such a true classification is not available in the case study that we discuss in Section 5.6. Since the Adjusted Rand Index cannot be used, we suggest to use the silhouette width instead. The silhouette width (SW) and the Adjusted Rand Index (ARI) appear to be positively and linearly related; see Figure 5.4. The linear regression explaining the silhouette width with the Adjusted Rand Index achieves an R^2 of 0.861, meaning that 86% of the variation in the Adjusted Rand Index values can be explained by changes in the silhouette width. The silhouette width appears to provide a worse estimation if the Adjusted Rand Index is either very low or very high. Nevertheless, the result indicates that, when the true clustering is not available and the Adjusted Rand Index cannot be used, the silhouette width by Rousseeuw (1987) is a reasonable alternative.

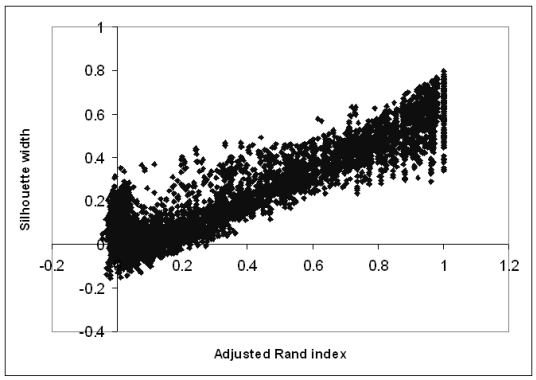

Figure 5.4. Adjusted Rand Index and silhouette width: a comparison

We explore two sources of variation in the results: the *spatial contiguity* and the *number of clusters*. *Spatial contiguity* is the degree to which geographically close subjects have a mutual influence on each other's attribute scores; see, for example, Ter Hofstede *et al.* (2002) and the references therein. We include spatial contiguity in our models as follows. When spatial contiguity is present in the data, subjects in the same segment are likely to be located close to each other. A seed point is determined for each segment. From this seed point, a path through the 100 by 100 plane is formed by adding random numbers drawn from $U[-a, a]$ to the current x and y-coordinates on which the subjects in the segment are placed, where the parameter $a \in \mathbb{N}$ determines the level of spatial contiguity. When the path reaches the boundary of the plane, the orientation is reversed. If the value of a is small, subjects in the same segment are located close to each

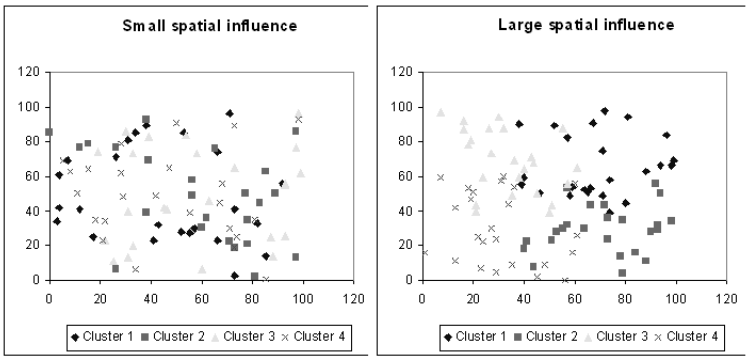

Figure 5.5. Subjects with small and large degrees of spatial contiguity

other. On the other hand, if the value of a is large, the subjects in the same segment are more or less randomly dispersed in the plane. Figure 5.5 shows typical planes with small and large simulated spatial contiguities. In case of large spatial contiguity, the subjects in the same cluster are located close to each other in the plane.

Figure 5.6 presents the effects of varying spatial contiguity. When the budget B is small, the fit of the cluster solution remains relatively high when the consumer preferences show a high degree of spatial contiguity.

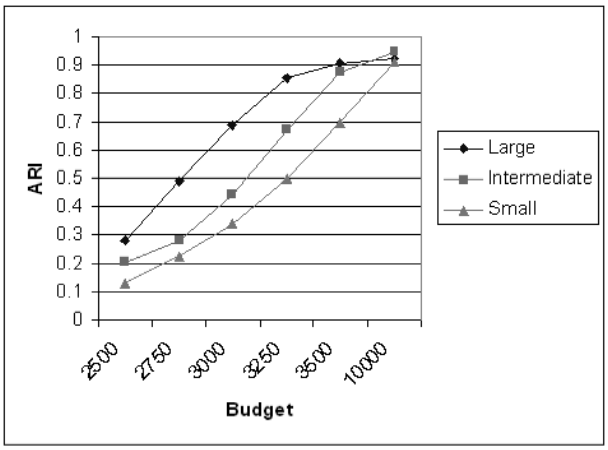

Figure 5.6. Average cluster solution quality for large, medium and small degrees of spatial contiguity

The second factor taken into account is the *number of clusters*. The number

of clusters in the data set of Milligan (1985) varies between 2 and 5. Figure 5.7 shows the quality of the solutions at different values of the budget B. The solution value, measured with the Adjusted Rand Index (ARI), decreases slightly with B when the number of segments is relatively large, but it declines strongly when $K = 2$. Obviously, when there are only a few clusters, remote subjects are forced into a cluster with different other subjects in order to keep logistics costs low.

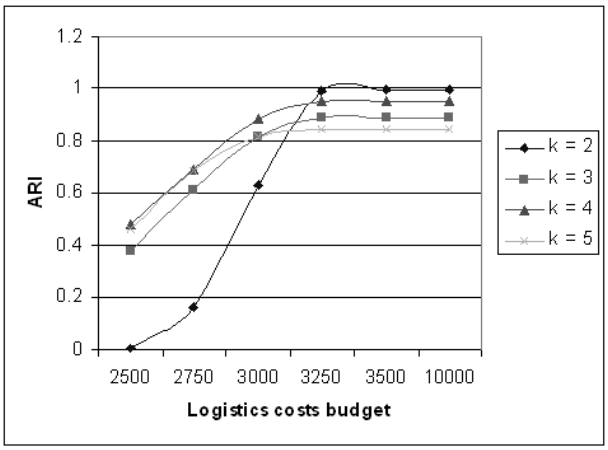

Figure 5.7. Average cluster solution quality and the number of clusters K

To summarize, the results of our experiments indicate that simulated annealing is a suitable method for solving clustering problems, in particular when a good starting solution is not available. Moreover, the Budget Constraint Approach presented in the previous section makes the trade-off between logistics costs and fit of segmentations. This trade-off is worthwhile when a decrease in logistics costs is associated with a slow decrease in fit of consumer preferences. This appears to occur most clearly for the random instances with standardized data from Milligan and Cooper (1986), when the number of segments should not be too small, and when spatial contiguity is present in the data.

5.5 Case study: European meat outlet segmentation

In this section, we describe a segmentation study on European meat outlets. The case study is used as an example of segmentation studies encountered by in-

ternational retail chains which adjust their stores to regional differences. The stores are tailored to the specific characteristics of the region in which the store is located. For each homogeneous group of regions, a separate retail formula is developed.

In order to determine how similar shops in different regions are, an international market research agency has executed a large survey on consumer behavior; see Ter Hofstede *et al.* (2002). Stores can obtain a distinctive position through the development of a particular *store image*. Important store image attributes are, for example, pricing, assortment, service, atmosphere, and quality. However, consumers in different regions may place a different relative importance on these attributes (Ter Hofstede *et al.*, 2002).

The results are obtained by sending mail questionnaires to members of a script panel in seven countries within the European Union, namely Germany, the Netherlands, Belgium, France, Spain, Portugal, and Italy. Store image measures are obtained, where for each respondent data are obtained for the store most frequently visited. The attributes are: price, quality, service, atmosphere, distance, and variety in meat. These six attributes are used to predict the general opinion about the store. For each of those six attributes and for the general opinion about the store, the respondent gives an opinion on a scale from 1 to 7. A separate regression is then carried out for each region in the data set, and the resulting regression parameters are used to estimate the relative importance of each attribute. If the number of respondents in a region is too small for a viable regression, the region borrows its characteristics from neighboring regions. The subjects of the segmentation are 123 so-called NUTS-2 regions, depicted in Figure 5.8. In order to compute the geographical distances in the case study, we use central points of regions, the *centroids*.

In Ter Hofstede *et al.* (2002), several mixture modeling approaches are implemented and compared. It is found that the so-called *spatial contiguity* model achieves the best fit of the data. In this model, the probability p_{ik} of region i belonging to segment k is not only influenced by the neighboring regions, but also by regions further away. The influence diminishes as the distance between a pair of regions is larger.

Logistics costs can play a key part in store segmentations. If the locations of shops within a selected segment are very dispersed, then supplying these shops

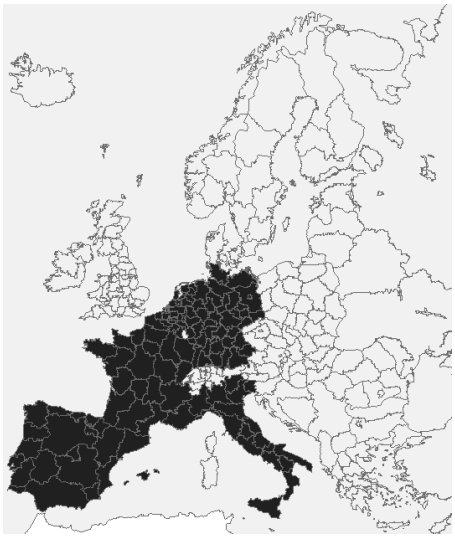

Figure 5.8. European regions in the data set

becomes expensive. Ter Hofstede *et al.* (2002) obtain clusters of regions that may be connected, but still stretch out over large sections of Europe, so with high logistics costs.

The mixture modeling approach is a suitable tool to determine what types of consumers are living in a region, and then offer multiple marketing mixes in each region in order to serve these segments. On the other hand, when only one retail formula can be offered in each region and an undifferentiated marketing approach within each region is chosen, Boone and Roehm (2002) find that hard clustering approaches, which assign a subject to a single segment, are more appropriate.

5.6 Computational experiments on the case study data

In this section, we construct segmentations for the case study discussed in the previous section. We use the Budget Constraint Approach from Section 5.3. We compare the resulting strategies with a mixture modeling approach, an unconstrained hard clustering approach, and the countries-as-segments segmentation strategy. Recall that the segmentation based are perceptions on price, quality, service, atmosphere, distance, and variety in meat.

The segmentations are evaluated with the following criteria:

Customer heterogeneity. The fit of customer heterogeneity, formed by the homogeneity of consumers within and the heterogeneity between segments, is measured with the Minimum Sum-of-Squares Criterium and the silhouette width; see Section 5.3. The MSSC is the optimization criterion for our hard cluster algorithm. Obviously, these methods are likely to achieve higher scores than mixture models, which maximize the classification likelihood. When a likelihood measure is taken instead, mixture models perform best; see e.g. Ter Hofstede *et al.* (1999). The silhouette width is a more neutral measure, similar to the Adjusted Rand Index from Section 5.4.

Number of segments. If possible, the number of segments in a segmentation should be kept small. For a large number of segments, many facilities need to be set up and new marketing strategies must be designed.

Unique characteristics of segments. For every segmentation solution, the attribute scores of all segments are evaluated. A segmentation performs well if each segment achieves a high score on one or more unique attributes (Ter Hofstede *et al.*, 1999). These unique attributes can be used to target the segments. For example, it is relatively easy to develop a marketing strategy for a segment that is very sensitive to price changes, or for a segment that appreciates quality highly.

Logistics costs. The logistics costs of each segmentation are estimated with the Distance to Central Facility (DCF) measure from Section 5.2. The sum of the distances from each region to its central segment facility is computed.

The first three criteria are also used in Ter Hofstede *et al.* (1999),where k-means and mixture modeling are compared.

Our first benchmark segmentation takes the countries as segments. Belgium and the Netherlands are joined into one segment, as well as Spain and Portugal. France, Germany and Italy are served as separate segments. The MSSC score of this segmentation is 22.96; the silhouette width is -0.0009. This means that much improvement in the solution quality can be achieved by transferring regions to

other segments. On the other hand, the logistics costs, measured as the sum of the distances of each region to its fictitious central segment facility, amount to 26813 km.

The second method is the standard hard clustering approach, consisting of a hierarchical method in the first stage and a non-hierarchical method in the second stage. Ward's method (Ward, 1963) is used to obtain a good initial solution and the number of clusters, namely $K = 4$. The initial solution is improved using simulated annealing (Kirkpatrick *et al.*, 1983), which we choose instead of the usual k-means approach. We obtain the solution depicted in Figure 5.9; the value of the MSSC is 12.16, and the DCF measure of logistics costs is 71552 km. The silhouette width score is 0.409. The longest distance between two regions in a single segment is 2326 km between the centroids of the regions Mecklenburg-Vorpommern in Northeast Germany and Algarve. This is the longest possible distance between any pair of regions in the data set.

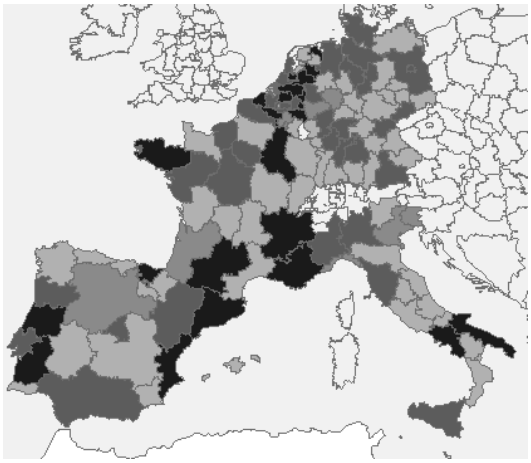

Figure 5.9. Solution obtained with unconstrained hard clustering

The *mixture modeling* is performed in Latent Gold (Vermunt and Magidson, 2003). The segments are estimated with normal distributions with different averages, but equal standard deviations. The number of segments in mixture modeling is usually determined with the AIC3 criterion; see Andrews and Currim (2003). In our case, the best fitting segmentation is obtained for $K = 4$. The posterior probabilities are rounded off, meaning that each region i is assigned to a retail formula k for which the probability p_{ik} is the highest. This

is a heuristic method, but the highest posterior probabilities are close to 1 for most regions. The resulting segmentation, depicted in Figure 5.10, has an MSSC score of 23.56, and the total distance is 68622 km. The silhouette width score is -0.058, which is even worse than the score of the countries-as-segments strategy. The mixture model leads to a very different segmentation from the unconstrained cluster solution and it achieves a very poor score on our fit criteria. This finding confirms the results from Boone and Roehm (2002).

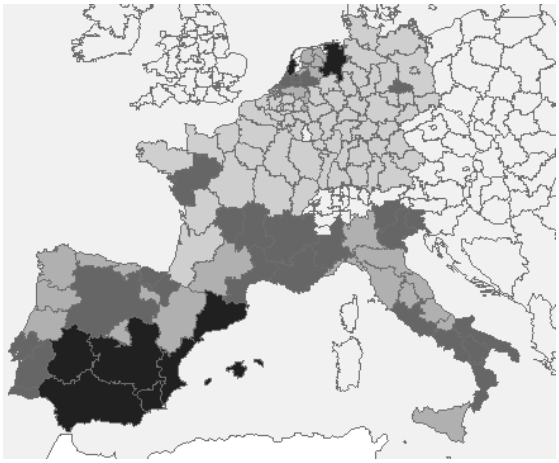

Figure 5.10. Solution obtained with mixture modelling

Next, we apply the Budget Constraint Approach from Section 5.3. The silhouette widths for segmentations with several values of B and K are shown in the surface plot from Figure 5.11 and in Table 5.4. The results indicate that the fit remains on a high level for smaller B vales when the value of K is large, namely $K = 6$ and $K = 7$.

In this segmentation study, the four segmentation strategies reported in Table 5.5 are efficient; see Section 5.3. If the logistics costs are the main determinant profits, then the countries-as-segments strategies is optimal. When the importance weights of the logistics costs w is increased, the segmentation with relatively low importance of the logistics costs consists of seven segments. This is option 1, and it is presented in Figure 5.12. The MSSC score is 9.52, the silhouette width is 0.433 and the logistics costs amount to 64219.

When w is increased, option 2 becomes the best alternative, with $K = 6$ and $B = 55000$. Option 2 combines large parts of Spain and Portugal into a

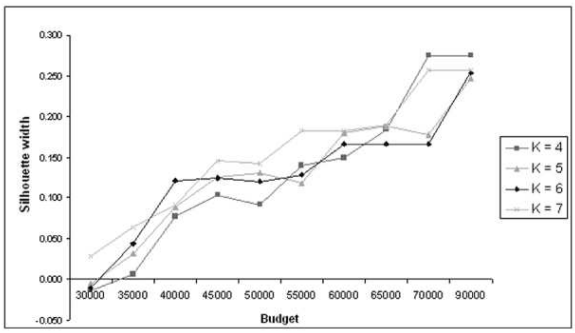

Figure 5.11. Fit of segmentations for various K and B

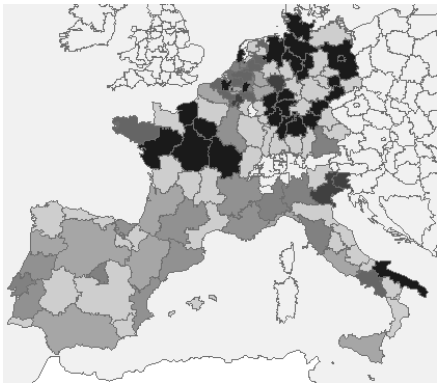

Figure 5.12. Segmentation option 1

Table 5.4. Silhouette width of segmentations for various values of K and B

	Number of clusters				
B	4	5	6	7	Best K
30000	0.006	0.028	-0.007	0.028	$K = 5$
35000	0.047	0.017	0.008	0.087	$K = 7$
40000	0.067	0.097	0.169	0.146	$K = 6$
45000	0.144	0.177	0.175	0.179	$K = 7$
50000	0.173	0.232	0.142	0.253	$K = 7$
55000	0.183	0.194	0.341	0.253	$K = 6$
60000	0.074	0.194	0.341	0.351	$K = 7$
65000	0.231	0.310	0.341	0.433	$K = 7$
70000	0.393	0.432	0.391	0.403	$K = 5$
Unconstr.	0.409	0.432	0.391	0.403	$K = 5$

Table 5.5. Segmentation alternatives based on silhouette widths

Option #	K	B	MSSC score	Silh. width	Log. cost	Relative weight log. costs
Option 1	7	65000	9.53	0.4329	64219	$w < 9.35$
Option 2	6	55000	11.40	0.3411	54399	$9.35 \leq w \leq 11.53$
Option 3	6	40000	14.47	0.029	39438	$11.53 \leq w \leq 14.11$
Country/segment	5	-	22.96	-0.009	26813	$w > 14.11$

single segment. The regions in this area are sufficiently similar to be combined into separate segments without losing much fit. The largest distance between two regions in the same segment is 1429 kilometers, the logistics costs amount to 54399, the MSSC score is 11.40, and the silhouette width is 0.341.

Option 3 also takes Southern Italy into a separate segment. In France, Germany and the low countries, the deviation from the traditional countries-as-segments is larger. The MSSC score is 14.47 and the silhouette width is 0.029. The logistics costs level is 39438.

Figure 5.15 summarizes the scores of the segmentations on the fit and the logistics costs. The fit of the segmentation is measured with the silhouette width and the logistics costs are measured with the distance to central location-measure. A fictitious ideal segmentation is located close to the right bottom corner of the figure.

Segmentations are also evaluated on the degree to which a segment has unique high scores on one or more characteristics. In Table 5.6, the segments averages of all segmentations are reported. In the hard unconstrained segmentation and in option 1, most segments are clearly reachable by one or more character-

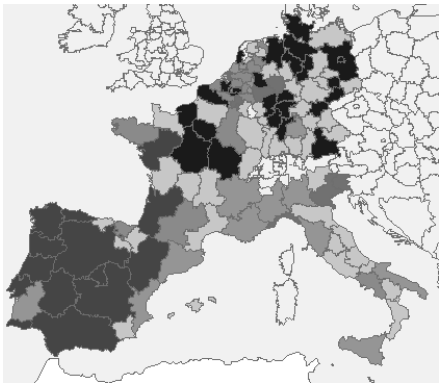

Figure 5.13. Segmentation option 2

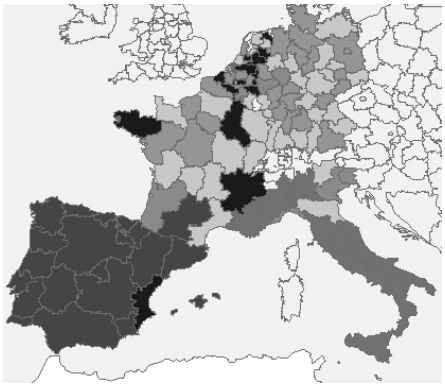

Figure 5.14. Segmentation option 3

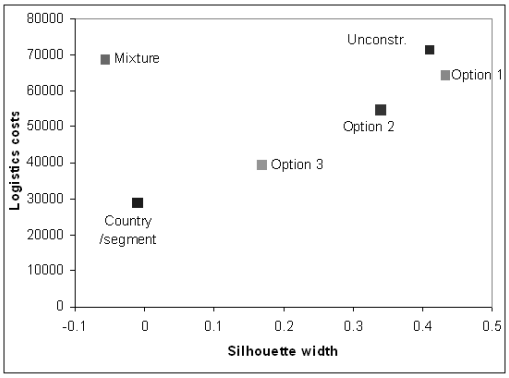

Figure 5.15. Comparison of logistics costs and fit of segmentation strategies

Table 5.6. Segment averages of different segmentations

	Attributes					
Segment	Price	Quality	Service	Atmosphere	Variety	Distance
			Countries as segments			
1	0.0360	0.1218	0.2963	0.1739	0.0850	0.2341
2	0.1047	0.1622	0.5319	0.0492	0.0747	0.1932
3	-0.0809	0.0191	0.2612	0.4240	0.1221	0.1142
4	-0.0272	0.4687	0.1724	0.3298	-0.0252	0.0301
5	-0.3024	0.1602	0.0485	0.5313	-0.0251	0.3451
			Hard unconstrained segmentation (option 1)			
1	0.0431	0.0404	0.0461	0.0516	0.0445	0.0427
2	-0.0006	0.1066	-0.3268	0.7601	0.0350	0.2379
3	0.0985	0.2080	0.1883	0.2614	0.1198	0.2431
4	-0.0040	0.0195	0.5395	0.0935	0.0605	0.1830
			Mixture model segmentation			
1	0.0598	0.1043	0.1668	0.1890	0.0638	0.1187
2	0.0401	0.0635	0.1296	0.1516	0.0689	0.1280
3	-0.0091	0.0518	0.1653	0.1007	0.0509	0.1816
4	0.0960	0.1531	0.1479	0.2123	0.1002	0.1713
			Option 1: segmentation with high logistics costs			
1	0.0362	0.0362	0.0362	0.0363	0.0362	0.0362
2	-0.0460	0.1777	-0.0605	0.3615	0.0905	0.3664
3	0.0279	-0.0969	0.6283	0.2003	0.0561	0.1439
4	-0.0998	0.0726	0.2607	0.3301	0.0228	0.1677
5	0.0238	0.1453	0.5069	-0.0661	0.0529	0.2563
6	0.0410	0.0390	-0.4103	0.9893	-0.0498	0.1195
7	0.1817	0.2047	0.2284	0.2175	0.1630	0.1725
			Option 2: segmentation with intermediate logistics costs			
1	0.0318	0.0396	0.0383	0.0414	0.0376	0.0416
2	-0.0309	-0.0785	0.4075	0.3493	0.0424	0.1842
3	0.0664	0.1475	0.5400	-0.0289	0.0751	0.1649
4	0.0022	0.0782	-0.3320	0.8355	-0.0348	0.2152
5	-0.0115	0.1523	0.0207	0.2154	0.0959	0.1821
6	0.1630	0.2213	0.1931	0.2171	0.1496	0.2408
			Option 3: segmentation with low logistics costs			
1	0.0410	0.0398	0.0437	0.0512	0.0440	0.0473
2	-0.0360	0.0138	0.4967	0.1043	0.0428	0.2359
3	0.1592	0.2207	0.2040	0.2209	0.1533	0.1956
4	0.0133	0.0382	0.1669	0.1740	0.0224	0.1228
5	0.0009	0.0822	0.1656	0.1384	0.0498	0.1194
6	-0.0040	0.0884	-0.3404	0.7703	0.0346	0.2587

istics. For example, segment 6 in option 1 values 'atmosphere' very high, and segment 4 from the unconstrained segmentation is very sensitive to service. In option 3, the segments 4 and 5 achieve similar scores, but segment 4 is located in Southern Italy, and segment 5 in Spain and Portugal. The difference in location justifies different retail formulas. The differences between the characteristics of the segments are smaller for mixture modeling than for hard clustering. This result implies that the targeting of segments is difficult when the mixture modeling segmentation strategy is chosen.

The Budget Constraint Approach provides the user with intermediate solutions with moderate logistics costs and a reasonable fit of consumer preferences, in addition to the countries-as-segments segmentation and the unconstrained segmentation. The consequence is that transnational segments can be served with an international segmentation with moderate logistics costs.

5.7 Reliability of simulated annealing

In Section 5.3, a simulated annealing approach for clustering is presented, and in Section 5.4 and 5.6, the approach is used to solve market segmentation problems with promising results. However, simulated annealing is a *randomized* search process: a deteriorating move to a new solution is made with probability $p(t)$, depending on the current temperature t, and the selection of the next trial move is done randomly. The final outcome depends on the realization of random variables, meaning that different cluster solutions may be obtained in different runs. In this section, we measure the variability in the quality of the resulting cluster solutions. If the variability is large, multiple runs should be carried out in order to rule out the possibility that, after an unlucky run, a low quality cluster solution is accepted. In statistics of quality, *repeatability* is the variability in the performance of a system due to internal factors of that system (Montgomery, 1997). We define repeatability here to be the degree under which an algorithm produces the same outcome of a cluster instance, given a set of fixed parameter values. An algorithm with a high degree of repeatability produces roughly the same solution in each run.

In the following experiment, the seed of the random distribution is varied; 200 different randomly generated seeds are taken. The seed determines the ac-

ceptance probabilities in the successive steps of the algorithm. The parameters of the simulated annealing and the starting solutions are fixed; the starting solution is the countries-as-segments solution from Section 5.6. The distribution of the MSSC scores of the resulting 200 simulated annealing solutions are depicted in Figure 5.16. After 200 runs, seven different simulated annealing solutions have been returned. Among them is an outlier with an MSSC score of 9.804 occurring in one run; we have no explanation for this outlier. The MSSC scores of all other solutions are situated in the interval $[9.18, 9.36]$. The remarkable outlier in the MSSC scores is not present in the the scores of the silhouette width.

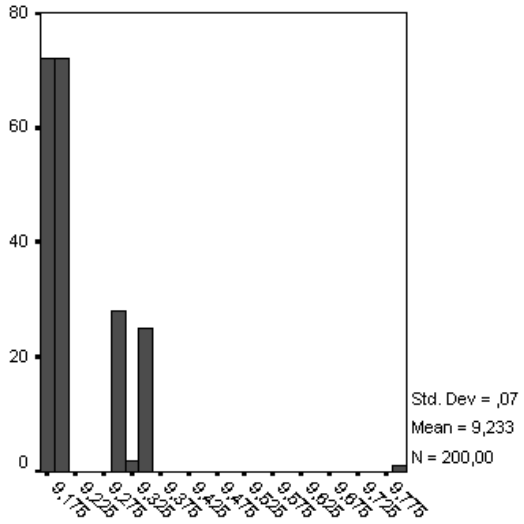

Figure 5.16. Histogram of MSSC scores

Table 5.7. Statistics of multiple simulated annealing

	MSSC	Silh. width	Log. costs
Mean	9.233	0.4558	70026
St. dev.	0.073	0.0045	417
Minimum	9.181	0.4454	68819
Maximum	9.804	0.4602	70902

Table 5.7 presents the most relevant statistics on the MSSC score, the silhouette width, and the logistics costs of the segmentation. The results show that

the variability between the solutions is small. Our results here indicate that simulated annealing is reliable when applied to the data set of Section 5.6, and a single run is probably sufficient.

5.8 An Alternative Approach: Restricting the Maximum Diameter within Segments

In the Budget Constraint Approach, the logistics costs of each candidate segmentation are computed. This can be a very time-consuming activity, in particular for large data sets. As a quick alternative, we present the *Diameter Constraint Approach*.

Low logistics costs are achieved when all consumers are easily and quickly reachable from a central facility, and the distance between each pair of consumers is small. An intuitive idea is to require segments to be connected; see e.g. Pawitan and Huang (2003) on Irish precipitation data. However, connected segments are no guarantee for low logistics costs. Instead, we choose to restrict the maximum distance between each pair of subjects in the same segment. This makes sense if there is a positive relationship between the logistics costs of a segment and the largest distance between two consumers a segment. More formally, a pair of subjects is allowed to be in the same segment if the geographical distance between them is at most D. We call D the *diameter* of the segment.

Is in true in general that a smaller diameter leads to segmentations with lower logistics costs? Our experimental results on the randomly generated instances from Milligan (1985), where the locations of the consumers are located on a 100 by 100 plane, shows that it is indeed true. Figure 5.17 shows the effect of changing the value of D on the logistics costs, measured with the DCF-measure. The logistics costs of the unconstrained segmentation are set to 100%. It appears that there is a positive relationship, but when the value of D increases beyond a threshold value, in this figure approximately $D = 110$, the logistics costs seem to stabilize. The explanation for this stabilization is that the resulting segmentations for values of D between 110 and 140 are more or less the same.

The *Diameter Constraint Approach* works along similar lines as the Budget Constraint Approach: it starts with a low logistics costs segmentation with a small value of D. Then it gradually increases the value of D until its value is

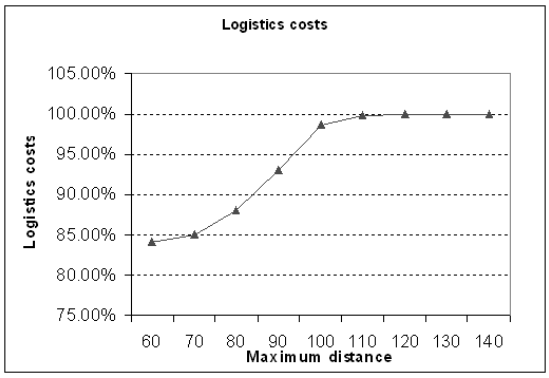

Figure 5.17. Logistics costs of random instances (unconstrained = 100%)

so large that no constraints are actually imposed on segmentations. As a consequence, we obtain segmentations with various diameters, and hence, with varying logistics costs.

The following clustering problem is obtained for selected combinations of K and D.

$$\begin{aligned}
\min \quad & \textstyle\sum_{i=1}^{N} \sum_{k=1}^{K} x_{ik} \|f_i - z_k\|^2 && && (5.8.1)\\
s.t. \quad & \textstyle\sum_{k=1}^{K} x_{ik} = 1 && i = 1, \ldots, N && (5.8.2)\\
& d_{ij} x_{ik} x_{jk} \leq D && i, j = 1, \ldots, N && \\
& && k = 1, \ldots, K && (5.8.3)\\
& x_{ik} \in \{0, 1\} && i = 1, \ldots, N && \\
& && k = 1, \ldots, K && (5.8.4)
\end{aligned}$$

Constraint (5.8.3) requires that the distance between two subjects in the same segment does not exceed D.

Consider the following small example with 15 consumers, and only one attribute on which the consumer scores are measured. Consumers 1 to 5 achieve a score of 2 on the attribute; 6 to 10 achieve a score of 6, and consumers 11 to 15 a score of 10. The consumers are randomly dispersed across a 20 by 20 plane; see Figure 5.18. The usual objective of segmentation is to construct segments which are homogeneous within and heterogeneous between segments. Clearly, the best segmentation divides the population into three segments consisting of the consumers 1 to 5, 6 to 10, and 11 to 15, respectively. Segment 1, indicated by

the triangles, and segment 3, indicated by the squares, have large geographical dispersions; see Figure 5.18.

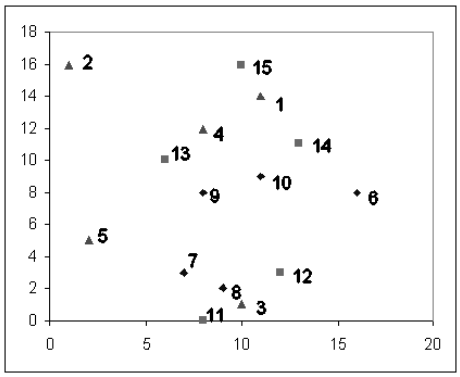

Figure 5.18. Location of consumers in the example

We reduce the logistics costs by restricting the diameter D of the segments to 15 and 10, respectively. When $D = 15$, consumer 4 moves from segment 1 to 2, and consumer 11 moves from segment 3 to 2, so the geographical outliers in segments 1 and 3 are reassigned. When D is decreased further to 10, three other consumers change segment membership; see Figure 5.19. The logistics costs of the segmentation decrease from 38.52 to 33.36 when $D = 15$ and to 27.96 when $D = 10$. On the other hand, when $D = 15$, two consumers are incorrectly classified, and when $D = 10$, the number of incorrectly classified consumers is five. This means that five consumers are served with marketing mixes which are ill-suited to their preferences. For this small example, a decrease in the maximum diameter leads to segmentations with lower logistics costs, but also to consumers being assigned to the wrong segment.

The choice of restricting the diameter D instead of B has two advantages. In the first place, it takes a much smaller amount of time to check whether each segment has diameter D than to check whether the logistics costs of a segmentation stay within the budget B. For example, in the case study presented in Section 5.5 ($N = 123$), the Diameter Constraint Approach needs only about two seconds to compute a good segmentation on a Pentium computer with speed 2 GHZ and 256 MB RAM under Windows 2000. On the other hand, the Budget Constraint Approach needs about 200 seconds on the same machine to complete the computation of a good segmentation for fixed values of K and B. The difference in

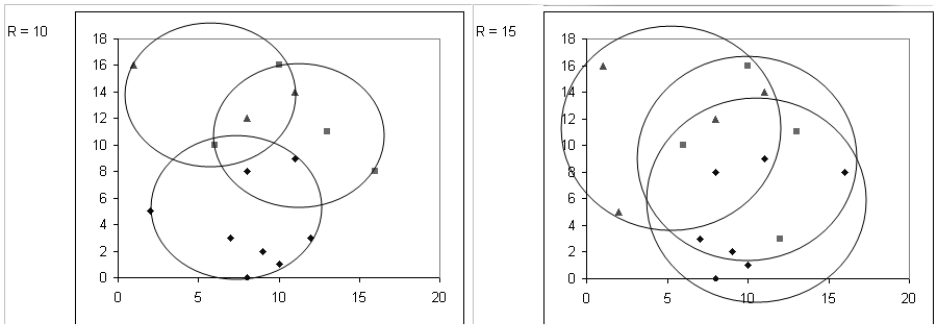

Figure 5.19. Segmentations with $D = 10$ (left) and $D = 15$ (right)

solution times becomes more evident for large data sets. Secondly, the diameter constraint can be implemented easily in the NORMCLUS of DeSarbo and Grisaffe (1998). This framework already contains constraints on the maximum travel distance between consumers.

A possible drawback of the Diameter Constraint Approach is that the diameter constraint applies to each individual segment. As a consequence, it is not possible to compensate a high logistics costs segment with a low logistics costs segment. This trade-off is clearly possible within the Budget Constraint Approach, so that the set of possible solutions is larger for the Budget Constraint Approach than for the Diameter Constraint Approach. At the same logistics costs level, we expect that the Budget Constraint Approach achieves a better fit score.

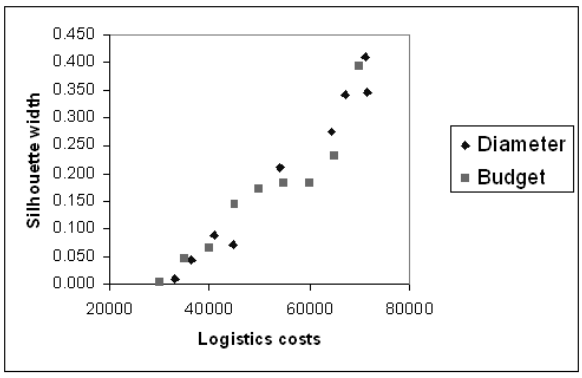

Figure 5.20. Comparison between BCA and DCA

Figure 5.20 shows the quality of the segmentations obtained with both approaches for $K = 4$; similar patterns are obtained for other K. The fit is measured with the silhouette width. The Diameter Constraint Approach achieves segmentations of approximately the same quality at fixed levels of the logistics costs, so it provides a reasonable approximation in case the Budget Constraint Approach is too time-consuming.

5.9 Solving Clustering Problems with Branch and Bound

Segmentation decisions are usually strategic decisions, and resources are fixed for long periods of time. For such decisions, it typically pays off to invest additional time to obtain optimal solutions. Here, we concentrate on exact methods for Minimum Sum-of-Squares Criterion (MSSC) clustering problems; see Section 5.3. Algorithms for other types of $\mathcal{NP}$-hard clustering problems are reviewed in Hansen and Jaumard (1997).

Most methods used to solve MSSC clustering problems are heuristics. BnB methods are presented in Koontz *et al.* (1975), Diehr (1985), and Du Merle *et al.* (2000). Exact methods are not so common for clustering problems, because until now, they have only been able to solve small instances to optimality. For example, the largest instance solved to optimality in Diehr (1985) contains 120 subjects, and in Du Merle *et al.* (2000) 150 subjects, requiring several hours of computing time. Exact methods for MSSC clustering are therefore only casually mentioned in the reviews by Jain *et al.* (1999) and Wedel and Kamakura (1998).

In Diehr (1985), a combinatorial BnB approach is presented that is based on the BnB algorithm from Koontz *et al.* (1975). In Du Merle *et al.* (2000), the MSSC clustering problem is formulated and solved as an Integer Programming problem. Variable Neighborhood Search (VNS), a meta-heuristic, is used to determine an initial solution. To the best of our knowledge, there is no direct comparison between both BnB algorithms, but the algorithm from Du Merle *et al.* (2000) solves harder and larger cluster instances to optimality.

The BnB algorithm from Diehr (1985) starts off by constructing an initial solution with a hill-climbing heuristic. It assigns subject 1 to cluster 1, leaving all other subjects unassigned. It then assigns subjects to clusters one by one in a prespecified order. The branching rule assigns the subject to the closest currently

available cluster. By 'closest' we mean that the center of the cluster is the closest to the attribute scores of the subject. Assume that there are N subjects to be clustered. When the BnB algorithm arrives at the n-th subject, it has already assigned and fixed the previous $n-1$ subjects to clusters. In Koontz *et al.* (1975), it is shown that the costs after n subjects have been assigned, form a lower bound for every cluster solution of that subproblem. In Diehr (1985), a tighter lower bound is suggested. When the BnB algorithm arrives at the n-th node, there are $N-n$ unassigned subjects left ($n = 1, \ldots, N-1$). In the terminology of Section 3.6, these subjects are called *offenders*, because the subjects are still unassigned. At subject n, a lower estimation of the minimum cost of assigning the remaining subjects to clusters is added to the lower bound.

It is too time-consuming to compute a new lower bound at each node of the search tree. Instead, the BnB algorithm determines optimal cluster solutions of subsets before the actual BnB process begins. For example, for the subsets $\{1, \ldots, 15\}, \{16, \ldots, 30\}, \ldots$, optimal solutions are computed. Diehr (1985) shows that the sum of the MSSC costs of optimal cluster solutions of subsets of subjects form a lower bound for the value of optimal cluster solutions. Small clustering problems of size 15 are solved rapidly. To obtain a lower bound, we first compute the cluster solutions of the subsets $\{N-14, \ldots, N\}, \{N-29, \ldots, N-15\}, \ldots$. At $n = N-30$, the solution values of the subsets $\{N-29, \ldots, N-15\}$ can be added to the current lower bound.

In this section, we take the simulated annealing solutions from Section 5.6, and try to improve these solutions using a BnB algorithm or to prove their optimality. Cluster problems with a restricted diameter D are also solved. We expect that the BnB algorithms are most successful when the search space is restricted, since the number of feasible solutions is then much smaller.

In Table 5.8, the basic BnB algorithm from Koontz *et al.* (1975) is compared to a version of the algorithm that employs simulated annealing to obtain an upper bound (UB), a version which employs the lower bound described above (LB), and a version that employs both (UB & LB). The basic algorithm has no additional upper or lower bound computations. The values in the table are the MSSC scores obtained when the corresponding the BnB algorithm is terminated after at most one hour of computing time. The instances are from the case study from Section 5.5, the number of clusters here is 6. In addition, we solve instances with

Table 5.8. Comparison of cluster solutions of BnB cluster algorithms with new upper and lower bounds

	Added to BnB algorithm				
D	Sim. ann.	Basic	UB	LB	UB & LB
800	19.679	22.668	19.679	22.668	19.679
1000	15.747	22.668	15.747	22.668	15.747
1200	13.851	22.668	13.851	22.668	13.851
1400	13.284	22.668	13.284	22.668	13.284
1600	11.881	22.668	11.881	22.668	11.881
1800	10.748	22.668	10.748	12.732	10.748
2000	10.236	13.002	10.236	12.459	10.236
2200	9.902	12.621	9.902	12.219	9.902
2400	9.292	12.188	9.292	11.828	9.292
2600	9.237	12.188	9.237	11.828	9.237

various values of D. The number of possibilities in the restricted cluster problems is smaller. Hence, we expect cluster problems with small values of D to be more easily solvable with the BnB algorithms.

It turns out that none of the BnB algorithms in Table 5.8 are able to solve one of the tested instances within an hour; the simulated annealing starting solution is not improved and optimality of the simulated annealing algorithm is not shown. It turns also out that for restricted problems, the BnB algorithms do not perform well. A possible reason is that the BnB algorithms have trouble finding good upper bounds. The lower bound improves the quality of the solutions obtained, but the results indicate that the lower bounds require the most urgent improvements.

In the iterative patching procedure of Chapter 2, upper bounds are constructed at each node of the search tree. Is it worthwhile to use simulated annealing in a similar iterative fashion, i.e., to compute upper bounds at multiple nodes of the BnB search tree? Iterative simulated annealing is probably not leading to tighter lower bounds. We have found that simulated annealing is relatively independent of starting solution, and that multiple optimization runs are not leading to solutions of different quality.

In this section, we find that the BnB algorithm described in Diehr (1985) is not able to improve the simulated annealing solution for instances of the case study from Section 5.5. This confirms the general agreement that exact algo-

rithms are not sufficiently developed to be usable in practice. For future research, it is interesting to include the approach by Du Merle *et al.* (2000) into the comparison.

5.10 Limitations and future research

The Budget Constraint Approach determines segmentations for various levels of the logistics costs budget B and the number of segments K. The logistics costs and the fit are then weighted, and a small number of candidate segmentations is obtained. This approach has the following limitations:

- The approach presented in this paper is tailored to 'hard' segmentations in which every consumer is assigned to a single segment, but it is doubtful whether it works well for general segmentations. The majority of segmentation studies is best solved with mixture models (Wedel and Kamakura, 1998). An interesting direction of future research is to develop a mixture modeling approach which limits the logistics costs, for example, by offering marketing mixes only to a closely located set of consumers. The NORMCLUS framework of DeSarbo and Grisaffe (1998) also offers great possibilities for constrained fuzzy clustering. Mixture models with restrictions on the set of feasible segmentations are also discussed in Law *et al.* (2004) and Shental *et al.* (2004).

- The Budget Constraint Approach returns a small set of candidate segmentations, but the choice for the most profitable one should still be made manually; we cannot compute directly which one generates the highest profit for the organization. The most profitable segmentation can be determined accurately if the model is able to convert the expected consumer benefits of a segmentation into revenues for the company.

- We assume that the number of segments does not influence the logistics costs level. This assumption appears to be plausible, since we find that the number of segments has very little influence on the transportation costs. If the number of segments is taken into account and weighted separately, efficient combinations of three variables need to be determined. This requires *multi-criteria analysis*.

- The DCF measure of the logistics costs is based on the assumption that each segment is served from a separate facility. When the geographical dispersion of consumers in the same segment is large, it may be the most profitable strategy to build one joint central facility, such as an EDC, for all segments. The cost savings of joining segment facilities are not taken into account.

- The performance of simulated annealing depends strongly on the choice of the parameters (Henderson *et al.*, 2003). This means that, in order to obtain high quality segmentations, these parameters should be readjusted for each new instance. An interesting direction of future research is to develop an *adaptive simulated annealing algorithm* (Henderson *et al.*, 2003), which automatically adapts the settings of the algorithm to the segmentation instance at hand. In our experiments, the initial temperature and the freezing temperature appear to be the most important parameters; they should both be set proportional to the approximate MSSC score of the cluster solution.

- An emerging topic in marketing is *store location* (Achabal *et al.*, 1982; Craig *et al.*, 1984; Clarke *et al.*, 1997). It turns out to be one of the key factors for the success of the stores (Reinartz and Kumar, 1999; Kumar and Karande, 2000; Mittal *et al.*, 2004). When a retail organization plans to serve the customers in a region with differentiated marketing mixes, it is not only important to know the characteristics of the different types of customers, but also their distances to possible store locations. The travel times can be seen as the logistics costs made by the consumers. An interesting direction of future research is the application of our approach to the problem of assigning similar customers to stores in a region, while minimizing the travel costs made by the customers. A similar application is described in Yorke (2001) where leisure facilities in Britain are considered. These facilities should be adapted to the desires of children in the neighborhood, but travel times to the locations should be taken into account simultaneously.

5.11 Conclusions

In the literature on segmentation, it is assumed that profit is maximized when the fit of consumer preferences is maximized. However, the costs of physical distribution make up, on average, about 20% of the total cost of a product (Davis, 1990). Steenkamp and Ter Hofstede (2002) report that logistics costs force organizations to maintain a countries-as-segments strategy in retailing and in case of perishable goods. We introduce new segmentation strategies, which make the trade-off between the consumer benefits of a segmentation and the logistics costs of serving the segments possible. The *Budget Constraint Approach* uses simulated annealing to construct segmentations. The simulated annealing algorithm finds good segmentation solutions, independent of the chosen starting solution (Wedel and Kamakura, 1998).

Research on random instances indicates that the Budget Constraint Approach is applied successfully if the selected number of segments is not too small. We have also applied the approach on a European meat outlet study, in which retail formulas are assigned to European regions. The approach is able to generate intermediate solutions for which both the logistics costs and the fit of the segmentation are reasonable.

The method presented in this paper is applicable to a small subclass of segmentations, namely those in which the company assigns the consumers to segments. Interesting directions of future research are to include the logistics costs into general segmentations for which mixture models appear to be the most suitable modeling method. Finally, it may be worthwhile to quantify the relationship between profit of the organization and the fit of consumer preferences. When the profits of a segmentation strategy are known, an effective cost-benefit analysis can be made.

Chapter 6

Conclusions

6.1 Introduction

In this dissertation, we consider solution techniques for the so-called *Combinatorial Optimization Problems (COPs)*. Given a finite set of elements, a COP is the problem of finding a combination of the elements of a subset that satisfies an a priori objective; see Section 1.1. Here, a minimum cost or maximum profit solution, i.e., an optimal solution, has to be selected. Examples of COPs can be found in network routing and scheduling, but also in many other fields of research; the market segmentation problem from Chapter 5 is an application in marketing.

There is a broad class of COPs for which exhaustive search is sometimes needed to find an optimal solution. As a consequence, finding a solution to a reasonably sized instance may take an amount of time that goes beyond any boundaries. The challenge for researchers is to enlarge the set of problems that can be solved to optimality within acceptable time limits. This dissertation presents improvements for the most common type of exact algorithms, namely *Branch and Bound (BnB)*; see Section 1.5. BnB is a collection of solution techniques in which an optimal solution is systematically searched in a so-called *search tree*: the original problem is divided into smaller problems. Although BnB solves many COPs within acceptable time limits, there is still a large set of problems that cannot be solved to optimality at the moment. In this thesis, we propose modifications of the technique that makes it suitable to solve those large or difficult problem instances to optimality.

We summarize this thesis on BnB chapterwise below.

6.2 Summary

Chapter 2: Iterative Patching and the Asymmetric Traveling Salesman Problem

Although Branch and Bound (BnB) methods are among the most widely used techniques for solving hard problems, it is still a challenge to make these methods smarter. In this chapter, we investigate *iterative patching*, a technique in which a fixed patching procedure is applied at each node of the BnB search tree for the Asymmetric Traveling Salesman Problem (ATSP), the problem of finding a shortest tour through a given set of locations. Computational experiments show that iterative patching results in general in search trees that are smaller than the classical BnB trees, and that solution times are lower for usual random and sparse instances. Furthermore, it turns out that, on average, iterative patching with the Contract-or-Patch procedure from Glover *et al.* (2001) and the Karp-Steele procedure are the fastest, and that 'iterative' Modified Karp-Steele patching generates the smallest search trees.

Chapter 3: Tolerance-Based Branch and Bound Algorithms for the Asymmetric Traveling Salesman Problem

The selection of entries to be included/excluded in Branch and Bound algorithms is usually done on the base of cost values. We consider the class of Depth First Search (DFS) algorithms, and propose to use *upper tolerances* to guide the search for optimal solutions. In spite of the fact that it needs time to calculate tolerances, our computational experiments for Asymmetric Traveling Salesman Problem show that in most situations tolerance-based algorithms outperform cost-based algorithms. The solution time reductions are mainly caused by the fact that the branching process becomes much more effective, so that optimal solutions are found in an earlier stage of the branching process. The use of tolerances also reveals why the widely used choice for branching on a smallest cycle in assignment solutions is on average the most effective one. Moreover, it turns out that tolerance-based DFS algorithms solve difficult practical instances faster than the Best First Search algorithm from Carpaneto *et al.* (1995).

Chapter 4: Additional Topics on Tolerance-Based Algorithms

In Chapter 4, additional topics on tolerance-based BnB algorithms are studied. First of all, it is found that tolerance-based BnB algorithms for the ATSP are effective on randomly generated instances with a large number of intercity distances. This is a pleasant finding, since Zhang (1993) shows that these instances are, on average, relatively difficult to solve. It also turns out that hybrid BnB algorithms, in which the power of tolerances and costs are combined, are usually not faster than the original tolerance-based algorithms for the ATSP. Finally, we show that, under certain conditions, it is possible to use multiple upper tolerance values simultaneously in a lower bound for the Degree-Constrained Minimum Spanning Tree Problem (DCMSTP).

Chapter 5: Balancing the Logistics Costs and the Fit of Market Segments

Chapter 5 applies COP techniques to market segmentation. Segments are typically formed to serve distinct groups of consumers with tailored marketing mixes, in order to better fit their needs. In geographic segmentation applications, a company responds to geographical differences by, for example, offering localized products. Existing segmentation strategies generate segments in which the constituting elements are not necessarily geographically closely located. When a geographically dispersed segment is served with one marketing mix, the logistics costs are high due to high transportation costs and long lead times. As a consequence, decision makers sometimes use other segmentation criteria. For example, a retail chain may decide to serve each country with its own formula. Such a segmentation strategy suffers from the disadvantage that transnational segments are ignored. Moreover, the results are expected to be suboptimal in terms of meeting customer needs.

In this chapter, we develop a segmentation method that balances the fit to consumer needs and logistics costs. The solution approaches are illustrated by means of the problem of assigning a set of European regions to retail formulas, using store attribute preferences of consumers as a segmentation basis. Compared to other methods, such as k-means, mixture models, and the countries-as-segments approach, our approach results in transnational segments that combine

moderate logistics costs with a relatively high level of within-segment homogeneity. These results are confirmed in experiments on randomly generated experiments. A practical implication of our study is that transnational segments may become profitable in markets with high logistics costs.

6.3 Contributions

The contributions of this dissertation are listed below.

- Our Depth First Search (DFS) algorithms are comparable in performance to the state-of-the-art *CDT algorithm* from Carpaneto *et al.* (1995) for the thoroughly studied ATSP. Our algorithms even achieve better performance for a variety of difficult ATSP instances. The improvements that enable this performance are summarized below.

- Upper bounds of BnB algorithms for the ATSP are improved in Chapter 2 with *iterative patching*, a procedure for constructing good feasible solutions of the ATSP. Iterative patching procedures reduce the number of subproblems in the usual BnB algorithm needs, so that smaller computation times are needed.

- Chapter 3 introduces tolerance-based branching rules and lower bounds. The tolerance-based branching rules divide the solution space in such a way that good or even optimal solutions are obtained relatively early in the search process. A BnB algorithm with tolerance-based lower bounds is able to remove much more subproblems than a BnB algorithm without such a lower bound. We also find a *synergy effect* between tolerance-based lower bounds and branching rules: when used simultaneously, they produce search tree reductions that are larger than the sum of the reductions from the individual improvements.

- When an algorithm departs from an initial solution, some elements need to be preserved, the *survival set*, whereas others need to be discarded, the *extinction set*. How can we predict whether an element belongs to the survival or to the extinction set? We present two measures for the quality of

the predictions, the *adjusted Rand index* and the *fit of the logistic regression model*. Both measures are used in Chapter 3 to compare the quality of the predictions of arc costs and arc tolerances for the ATSP. The results are consistent with the actual reductions of the tolerance-based and the cost-based branching rules. The predictions with logistic regression appear to be a little better than those with the adjusted Rand index.

- In Chapter 4, we compare our best upper tolerance-based BnB algorithm with a cost-based BnB algorithm for randomly generated instances of the ATSP. It appears that the tolerance-based algorithm is particularly effective when there is a large number of different intercity distances. It is shown in Zhang (1993) that these random instances are on average difficult to solve.

- We provide a lower bound that adds multiple upper tolerance values to lower bounds. Such improved lower bounds may increase the speed of BnB algorithms, and provide better estimations on the accuracy of given heuristic solutions.

- Steenkamp and Ter Hofstede (2002) observe that, in many segmentations in practice, logistics costs are so prohibitive that regular segmentations are not profitable. Instead, the segmentation strategy with countries or regions as segments is chosen. These segmentations are often ill-suited to consumer preferences, which tend to have a transnational and interregional character. The Budget Constraint Approach, introduced in Chapter 5, makes the trade-off between the fit and the logistics costs of segmentations possible. It paves the road for interregional or international segments with moderate logistics costs and reasonable costs.

- In current literature, the concept of logistics costs of a segmentation has not been thoroughly considered yet. In Chapter 5, a model of the logistics costs of segmentations is given. The underlying assumption behind this model is that each location in a segment is supplied from a central facility. The model can easily be extended to decentral facilities.

- Our experiments in Chapter 5 show that *simulated annealing* is an effective and reliable meta-heuristic for obtaining cluster solutions. It outper-

forms BnB.

6.4 Limitations and future research

We conclude this dissertation with limitations and future research.

- This dissertation concentrates on BnB algorithms for the ATSP. The question is whether the iterative patching-like procedures from Section 2 and the tolerance-based branching rules and lower bounds from Section 3 and 4 can also be used for other COPs. This is an interesting direction of future research.

- Research here is restricted to DFS BnB algorithms: algorithms that solve the most recently generated subproblem in the BnB process first. In practice, BFS BnB algorithms are also frequently used. An interesting direction of future research is to include tolerance computations in BFS algorithms.

- In Chapter 2, we limit the upper bounding heuristics to four known patching algorithms. Other types of heuristics may be used as well. In particular, the application of meta-heuristics to obtain upper bounds for BnB algorithms appears to be interesting.

- Chapter 3 focuses on upper tolerances. Roughly spoken, the upper tolerance of an element is the increase in the cost value of an element needed to remove it from an optimal solution. It is also possible to consider lower tolerance of an element, which is, roughly spoken, the minimum decrease in the cost value of an element needed to include it in an optimal solution. So, instead of *removing* elements from a given optimal solution, elements from outside the optimal solution are *included*. Since there are usually more elements outside the optimal solution than inside, it appears that lower tolerance computations are too time-consuming. However, it is shown in Volgenant (2006) that this need not be true, and in Germs (2006), lower tolerance-based BnB algorithms prove to be about as fast as their upper tolerance-based counterparts for the ATSP. So lower tolerance based algorithms constitute a very promising field of research.

- Upper tolerances are used here only in BnB algorithms. They can also be applied to heuristics.
- We have tested the additive lower bound from Chapter 4 only on Symmetric TSP instances. The bound is also appropriate for the Degree Constrained Minimum Spanning Tree Problem (DCMSTP). Computational experiments should be conducted to prove the effectivity of the additive bound for the DCMSTP.
- The settings of the simulated annealing algorithm from Chapter 5 are determined empirically, i.e., for a given set of instances, the parameter values are determined that leads to the highest solution quality. An interesting direction of research is to make a 'general' simulated annealing algorithm that automatically adjust its parameter settings to properties of the instances at hand, such as the size of the instance or the type of attribute data.

Bibliography

Abrantes, A. and J. Marques (1996), A Class of Constrained Clustering Algorithms for Object Boundary Extraction, *IEEE Transactions on Image Processing*, **5**, 1507–1521.

Achabal, D., W. Gorr, and V. Mahajan (1982), MULTILOC: A Multiple Store Location Model, *Journal of Retailing*, **58**, 5–25.

Achterberg, T., T. Koch, and A. Martin (2004), Branching Rules Revisited, *Operations Research Letters*, **33**, 42–54.

Affenzeller, M. and R. Mayrhofer (2002), Generic Heuristics for Combinatorial Optimization Problems, `http://www.heuristiclab.com/publications/papers/affenzeller02c.pdf`.

Aho, A., J. Hopcroft, and J. Ullman (1974), *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA.

Allenby, G., N. Arora, and J. Ginter (1998), On the Heterogenity of Demand, *Journal of Marketing Research*, **XXXV**, 384–389.

Andrews, R. and I. Currim (2003), A Comparison of Segment Retention Criteria for Finite Mixture Logit Models, *Journal of Marketing Research*, **25**, 235–243.

Applegate, A., R. Bixby, V. Chvátal, W. Cook, and K. Helsgaun (2004), `http://www.tsp.gatech.edu/sweden/index.html`.

Balas, E. and P. Toth (1985), Branch and Bound Methods, in E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys (eds.), *The Traveling Salesman Problem*, John Wiley & Sons, Chichester, pp. 361–401.

Bang-Jensen, J. and G. Gutin (2001), *Digraphs: Theory, Algorithms and Applications*, Springer-Verlag, London.

Baptiste, P., J. Carlier, and A. Jouglet (2004), A Branch-and-Bound Procedure to Minimize Total Tardiness on One Machine with Arbitrary Release Dates, *European Journal of Operational Research*, **158**, 595–608.

Basseur, M., J. Lemesre, C. Dhaenens, and E.-G. Talbi (2004), Cooperations between Branch and Bound and Evolutionary Algorithms to Solve a Biobjective Flow Shop Problem, *Workshop on Experimental and Efficient Algorithms (WEA '04)*, 72–86.

Bijmolt, T., L. Paas, and J. Vermunt (2004), Country and Consumer Segmentation: Multi-level Latent Class Analysis of Financial Product Ownership, *International Journal of Research in Marketing*, **21**, 323–340.

Bixby, R. (1994), Progress in Linear Programming, *ORSA Journal on Computing*, **6**, 15–22.

Boone, D. and M. Roehm (2002), Retail Segmentation using Artificial Neural Networks, *International Journal of Research in Marketing*, **19**, 287–301.

Brusco, M., J. Cradit, and S. Stahl (2002), A Simulated Annealing Heuristic for a Bicriterion Partitioning Problem in Market Segmentation, *Journal of Marketing Research*, **39**, 99–109.

Buriol, L., P. França, and P. Moscato (2004), A New Memetic Algorithm for the Asymmetric Traveling Salesman Problem, *Journal of Heuristics*, **10**, 483–506.

Burkard, R. (1997), Efficiently Solvable Special Cases of Hard Combinatorial Optimization Problems, Tech. Rep. 105, Karl-Franzens-Universität Graz & Technische Universität Graz.

Caccetta, L. and S. Hill (2001), A Branch and Cut Method for the Degree-Constrained Minimum Spanning Tree Problem, *Networks*, **37**, 74–83.

Carpaneto, G., M. Dell'Amico, and P. Toth (1995), Exact Solution of Large-scale Asymmetric Traveling Salesman Problems, *ACM Transactions on Mathematical Software*, **21**, 394–409.

Carpaneto, G. and P. Toth (1980), Some New Branching and Bounding Criteria for the Asymmetric Traveling Salesman Problem, *Management Science*, **21**, 736–743.

Clarke, I., D. Bennison, and J. Pal (1997), Towards a Contemporary Perspective of Retail Location, *International Journal of Retail and Distribution Management*, **25**, 59–69.

Clay Mathematics Institute (2006), Millennium Problem Homepage, `http://www.claymath.org/millennium/`.

Climer, S. and W. Zhang (2006), Cut-and-Solve: an Iterative Search Strategy for Combinatorial Optimization Problems, *Artificial Intelligence*, **170**, 714–738.

Cook, S. (1971), The Complexity of Theorem-Proving Procedures, in *Conference Record of Third Annula ACM Symposum on Theory of Computing (3rd STOC, Shaker Heights, Ohio, 1971)*, The Asociation for Computing Machinery, New York, pp. 151–158.

Cook, W., W. Cunningham, W. Pulleyblank, and A. Schrijver (1998), *Combinatorial Optimization*, John Wiley & Sons, Chichester.

Cotta, C. and J. Troya (2003), Embedding Branch and Bound within Evolutionary Algorithms, *Applied Intelligence*, **18**, 137–153.

Craig, C., A. Ghosh, and S. McLafferty (1984), Models of the Retail Location Process: A Review, *Journal of Retailing*, **60**, 5–36.

Dantzig, G., A. Orden, and W. P. (1955), The Generalized Simplex Method for Minimizing a Linear Form under Inequality Restraints, *Pacific Journal of Mathematics*, **8**, 183–195.

Davis, H. (1990), Distribution Costs and Customer Service Level: How Do You Compare in 1990?, in *Proceedings Annual Conference*, Council of Logistics Management, Anaheim.

Dell'Amico, M. and P. Toth (2000), Algorithms and Codes for Dense Assignment Problems: the State of the Art, *Discrete Applied Mathematics*, **100**, 17–48.

DeSarbo, W. and D. Grisaffe (1998), Combinatorial Optimization Approaches to Constrained Market Segmentation: An Application to Industrial Market Segmentation, *Marketing Letters*, **9**, 115–134.

Diehr, G. (1985), Evaluation of a Branch and Bound Algorithm for Clustering, *SIAM Journal on Scientific and Statistical Computing*, **6**, 268–284.

Du Merle, O., P. Hansen, B. Jaumard, and N. Mladenovic (2000), An Interior Point Algorithm for Minimum Sum-of-Squares Clustering, *SIAM Journal of Computing*, **21**, 1485–1505.

Dunne, P., A. Gibbons, and M. Zito (2000), Complexity-theoretic Models of Phase Transitions in Computer Science, *Theoretical Computer Science*, **249**, 243–263.

Euro-CASE (2001), Freight Logistics and Transport Systems in Europe.

Everitt, B., M. Leese, and S. Landau (2001), *Cluster Analysis*, 4th edn., Oxford University Press.

Fischetti, M., A. Lodi, and P. Toth (2002), Exact Methods for the Asymmetric Traveling Salesman Problem, in G. Gutin and A. Punnen (eds.), *The Traveling Salesman Problem and its Variations*, Kluwer Academc Publishers, pp. 169–194.

Francis, R. and J. White (1974), *Facility Layout and Location: An Analytical Approach*, Prentice-Hall, Englewood Cliffs, NJ.

Garey, M. and D. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco.

Germs, R. (2006), *Lower Tolerance-Based Branch and Bound Algorithms for the ATSP*, Master's thesis, University of Groningen.

Gessner, G., N. Malhotra, W. Kamakura, and M. Zmijevski (1988), Estimating Models with Binary Dependent Variables: Some Theoretical and Empirical Observations, *Journal of Business Research*, **16**, 49–65.

Glover, F., G. Gutin, A. Yeo, and A. Zverovich (2001), Construction Heuristics and Domination Analysis for the Asymmetric Traveling Salesman Problem, *European Journal of Operational Research*, **129**, 555–568.

Glover, F. and G. Kochenberger (2003), *Handbook of Metaheuristics*, Kluwer Academic Publishers.

GMA (2005), *Transportation Costs Driving Up Production Costs, CPG Industry Survey Finds*.

Goldengorin, B. (2002), *Data Correcting Algorithms in Combinatorial Optimization*, Ph.D. thesis, University of Groningen.

Goldengorin, B., D. Ghosh, and G. Sierksma (2004), Branch and Peg Algorithms for the Simple Plant Location Problem, *Computers & Operations Research*, **31**, 241–255.

Goldengorin, B., G. Jäger, and P. Molitor (2006), Tolerances Applied in Combinatorial Optimization, *Journal of Computer Science*, **2**, 716–734.

González, J., I. Rojas, M. Salmerón, A. Prieto, and J. Merelo (2001), Optimization of Web Newspaper Layout in Real Time, *Computer Networks*, **36**, 3111–321.

Gutin, G. and A. Zverovich (2005), Evaluation of the Contract-or-Patch Heuristic for the Asymmetric Traveling Salesman Problem, *INFOR*, **43**, 23–31.

Hair, J., R. Anderson, R. Tatham, and W. Black (1998), *Multivariate Data Analysis*, 5th edn., Prentice-Hall.

Hansen, P. and B. Jaumard (1997), Cluster Analysis and Mathematical Programming, *Mathematical Programming*, **79**, 191–215.

Held, M. and R. Karp (1970), The Traveling Salesman Problem and Minimum Spanning Trees, *Operations Research*, **18**, 1138–1162.

Helsen, K., K. Jedidi, and W. DeSarbo (1993), A New Approach to Country Segmentation Utilizing Multinational Diffusion Patterns, *Journal of Marketing*, **57**, 60–71.

Helsgaun, K. (2000), An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, *European Journal of Operational Research*, **12**, 106–130.

Henderson, D., S. Jacobson, and A. Johnson (2003), The Theory and Practice of Simulated Annealing, in F. Glover and G. Kochenberger (eds.), *Handbook of Metaheuristics*, chap. 10, Kluwer.

Hogg, T. (1996), Refining the Phase Transition in Combinatorial Search, *Artificial Intelligence*, **81**, 127–154.

Hogg, T., B. Huberman, and C. Williams (1996), Phase Transitions and the Search Problem, *Artificial Intelligence*, **81**, 1–15.

Hosmer, D. and S. Lemeshow (1989), *Applied Logistic Regression*, 1st edn., John Wiley & Sons.

Hubert, L. and P. Arabie (1985), Comparing Partitions, *Journal of Classification*, **2**, 193–218.

Ibaraki, T. (1987), *Enumerative Approaches to Combinatorial Optimization*, vol. 10 of *Annals of Operations Research*, J.C. Baltzer AG.

Jain, A., M. Murty, and P. Flynn (1999), Data Clustering: A Review, *ACM Computing Surveys*, **31**, 264–323.

Johnson, D. (1991), A Theoretician's Guide to the Experimental Analysis of Algorithms, Tech. rep., AT & T Research Labs.

Johnson, D. (2006), Machine Comparison Site, `http://public.research.att.com/~dsj/chtsp/speeds.html`.

Johnson, D., G. Gutin, L. McGeoch, A. Yeo, W. Zhang, and A. Zverovich (2002), Experimental Analysis of Heuristics for the ATSP, in G. Gutin and A. Punnen (eds.), *The Traveling Salesman Problem and its Variations*, chap. 10, Kluwer, Dordrecht, pp. 445–489.

Jonker, R. and A. Volgenant (1986), Improving the Hungarian Assignment Algorithm, *Operations Research Letters*, **5**, 171–175.

Jungnickel, D. (2005), *Graphs, Networks and Algorithms*, 2nd revised edn., Springer Verlag, Berlin.

Kann, V. and P. Crescenzi (2006), A Compendium of NP Optimization Problems, `http://www.nada.kth.se/%7Eviggo/wwwcompendium/`.

Karp, R. (1972), Reducibility Among Combinatorial Problems, in *Complexity of Computer Computations, Porc. Sympos. IBM Thomeas J. Watson Research Center, Yorktown Heights, N.Y.*, Plenum.

Karp, R. (1975), On the Computational Complexity of Combinatorial Problems, *Networks*, **5**, 45–68.

Karp, R. (1979), A Patching Algorithm for the Nonsymmetric Traveling Salesman Problem, *SIAM Journal of Computing*, **8**, 561–573.

Karp, R. and J. Steele (1990), Probabilistic Analysis of Heuristics, in E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys (eds.), *The Traveling Salesman Problem*, Wiley, New York, pp. 181–205.

Kirkpatrick, S., C. Gelatt, and M. Vecchi (1983), Optimization by Simulated Annealing, *Science*, **220**, 671–680.

Klau, G., I. Ljubic, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. Reidl, and R. Weiskircher (2004), Combining a Memetic Algorithm with Integer Programming to Solve the Prize-collecting Steiner Tree Problem, in K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. Lanzi, I. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell (eds.), *Genetic and Evolutionary Computation - GECCO 2004, Lecture Notes in Computer Science 2103*, Springer Verlag, Berlin, pp. 1304–1315.

Klee, V. and G. Minty (1972), How Good is the Simplex Algorithm?, in O. Shisha (ed.), *Inequalitites, III*, Academic Press, New York, pp. 159–175.

Klein, R. and R. Dubes (1989), Experiments in Projection and Clustering by Simulated Annealing, *Pattern Recognition*, **22**, 213–220.

Knuth, D. (1997), *The Art of Computer Programming*, vol. 1, third edn., Addison-Wesley.

Koontz, W., P. Narendra, and K. Fukunaga (1975), A Branch and Bound Clustering Algorithm, *IEEE Transactions on Computers*, **24**, 908–915.

Kotler, P. and G. Armstrong (2006), *Principles of Marketing*, 11th edn., Prentice Hall.

Kotler, P., D. Haider, and I. Rein (1993), *Marketing Places*, Free Press, New York.

Krarup, J. and P. Pruzan (1989), Ingredients of Locational Analysis, in P. Mirchandani and R. Francis (eds.), *Discrete Location Theory*, chap. 1, Wiley, New York, pp. 1–55.

Krishnamoorthy, M., A. Ernst, and Y. Sharaiha (2001), Comparison of Algorithms for the Degree Constrained Minimum Spanning Tree Problem, *Journal of Heuristics*, **7**, 587–611.

Kumar, V. and K. Karande (2000), The Effect of Retail Store Environment on Retailer Performance, *Journal of Business Research*, **49**, 167–181.

Kumar, V. and J. Petersen (2005), Using a Customer-Level Marketing Strategy to Enhance Firm Performance: A Review of Theoretical and Empirical Evidence, *Journal of the Academy of Marketing Science*, **33**, 504–519.

Law, M., A. Topchy, and A. Jain (2004), Clustering with Soft and Group Constraints, in *Joint IAPR Workshop on Syntactical,Structural, and Statistical Pattern Recognition*.

Libura, M. (1991), Sensitivity Analysis for Minimum Hamiltonian Path and Traveling Salesman Problems, *Discrete Applied Mathematics*, **30**, 197–211.

Libura, M., E. Van der Poort, G. Sierksma, and J. Van der Veen (1998), Stability Aspects of the Traveling Salesman Problem Based on k-best Solutions, *Discrete Applied Mathematics*, **87**, 159–185.

Lin, C.-J. and U.-P. Wen (2003), Sensitivity Analysis of the Optimal Assignment, *Discrete Optimization*, **149**, 35–46.

Linderoth, J. and M. Savelsbergh (1997), A Computational Study of Search Strategies for Mixed Integer Programming, Tech. rep., Georgia Institute of Technology.

Loiola, E., M. De Abreu, P. Boaventura-Notto, P. Hahn, and T. Querido (2007), A Survey for the Quadratic Assignment Problem, *European Journal of Operational Research*, **176**, 657–690.

MacQueen, J. (1967), Some Methods for Classification and Analysis of Multivariate Observations, in *Proceedings of Fifith Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press.

Miller, D. and J. Pekny (1991), Exact Solution of Large Asymmetric Traveling Salesman Problems, *Science*, **251**, 754–761.

Milligan, G. (1985), An Algorithm for Generating Artificial Test Clusters, *Psychometrika*, **50**, 123–127.

Milligan, G. and M. Cooper (1986), A Study of the Comparability of External Criteria for Hierarchical Cluster Analysis, *Multivariate Behavioral Research*, **21**, 441–458.

Milligan, G. and M. Cooper (1987), Methodology Review: Clustering Methods, *Applied Psychological Measurement*, **11**, 329–354.

Mirchandani, P. and R. Francis (1989), *Discrete Location Theory*, Wiley New York.

Mirkin, B. and I. Muchnik (1998), Combinatorial Optimization in Clustering, in D. Du and P. Pardalos (eds.), *Handbook of Combinatorial Optimization, Volume 2*, Kluwer Academic Publishers, pp. 216–330.

Mittal, V., W. Kamakura, and R. Govind (2004), Geographic Patterns in Customer Service and Satisfaction: An Empirical Investigation, *Journal of Marketing*, **68**, 48–62.

Montgomery, D. (1997), *Introduction to Statistical Quality Control*, 3rd edn., Jon Wiley & Sons.

Moscato, P. and C. Cotta (2003), A Gentle Introduction to Memetic Algorithms, in F. Glover and G. Kochenberger (eds.), *Handbook of Metaheuristics*, Kluwer Acamdemic Publishers, pp. 105–144.

Naddef, D. (2002), Polyhedral Theory and Branch-and-Cut Algorithms for the Symmetric TSP, in G. Gutin and A. Punnen (eds.), *The Traveling Salesman Problem and its Variations*, Kluwer, Dordrecht, pp. 29–116.

Nelson, P. and G. Toledano (1979), Challenges for International Logistics Management, *Journal of Business Logistics*, **1**, 1–21.

Nwana, V., K. Darby-Nowman, and G. Mitra (2005), A Co-operative Parallel Heuristic for Mixed Zero-One Linear Programming: Combining Simulated Annealing with Branch and Bound, *European Journal of Operational Research*, **164**, 12–23.

Pastor, R. and A. Corominas (2004), Branch and Win: OR Tree Search Algorithms for Solving Combinatorial Optimisation Problems, *Top*, **1**, 169–192.

Pawitan, Y. and J. Huang (2003), Constrained Clustering of Irregularly Sampled Rainfall Data, *Journal of Statistical Computation and Simulation*, **73**, 853–865.

Prim, R. (1957), Shortest Connection Networks and Some Generalizations, *Bell System Technology Journal*, **36**, 1389–1401.

Punj, G. and D. Stewart (1983), Cluster Analysis in Marketing Research: Review and Suggestions for Application, *Journal of Marketing Research*, **20**, 134–148.

Punnen, A. (2002), *The Traveling Salesman Problem: Applications, Formulations and Variations*, chap. 1, Kluwer Academic Publishers, pp. 1–28.

Reinartz, W. and V. Kumar (1999), Store-, Market-, and Consumer-Characteristics: The Drivers of Store Performance, *Marketing Letters*, **10**, 5–22.

Reinelt, G. (1991), TSPLIB - a Traveling Salesman Problem Library, *ORSA Journal on Computing*, **3**, 376–384.

Reinelt, G. (1995), TSPLIB, `http://www.informatik.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

Rousseeuw, P. (1987), Silhouettes, a Graphical Aid to the Interpretation and Validation of Cluster Analysis, in *Journal of Computational and Applied Mathematics*, 20.

Salah, H. and L. Pervel Kathanis (1994), Global Market Segmentation and Trends, in E. Kaymek and H. Salah (eds.), *Globalization of Consumer Markets: Structures and Strategies*, International Business Press, New York, pp. 47–63.

Schrijver, A. (2003), *Combinatorial Optimization : Polyhedra and Efficiency*, Springer Verlag, Berlin.

Shental, N., A. Bar-Hillel, T. Hertz, and D. Weinshall (2004), Computing Gaussian mixture models with EM using equivalence constraints, in *Advances in Neural Information Processing Systems 16*, MIT Press.

Sierksma, G. (1996), *Integer and Linear Programming*, Marcel Dekker, New York.

Simonson, I. (2005), Determinants of Customers' Responses to Customized Offers: Conceptual Framework and Research Propositions, *Journal of Marketing*, **69**, 32–45.

Smith, W. (1956), Product Differentation and Market Segmentation as Alternative Marketing Strategies, *Journal of Marketing*, **21**, 3–8.

Steenkamp, J. and F. Ter Hofstede (2002), International Market Segmentation: Issues and Perspectives, *International Journal of Research in Marketing*, **19**, 185–213.

Ter Hofstede, F., J. Steenkamp, and M. Wedel (1999), International Market Segmentation Based on Consumer-Product Relations, *Journal of Marketing Research*, **36**, 1–17.

Ter Hofstede, F., M. Wedel, and J. Steenkamp (2002), Identifying Spatial Segments in International Markets, *Marketing Science*, **21**, 160–177.

Todd, M. (2002), The Many Facets of Linear Programming, *Mathematical Programming*, **91**, 417–436.

Van der Veen, J. (1992), *Solvable Cases of the Traveling Salesman Problem with Various Objective Functions*, Ph.D. thesis, University of Groningen.

Van Laarhoven, P. and E. Aarts (1987), *Simulated Annealing: Theory and Applications*, Mathematics and its Aplications, D. Reidel Publishing Company.

Vermunt, J. and J. Magidson (2003), Latent Class Models for Classification, *Computational Statistics and Data Analysis*, **41**, 531–537.

Volgenant, A. (1989), A Lagrangean Approach to the Degree-Constrained Minimum Spanning Tree Problem, *European Journal of Operational Research*, **39**, 325–331.

Volgenant, A. (2006), An Addendum on Sensitivity Analysis of the Optimal Assignment, *European Journal of Operational Research*, **169**, 338–339.

Ward, J. (1963), Hierarchical Grouping to Optimize an Objective Function, *Journal of the American Statistical Assciation*, **58**, 236–244.

Wedel, M. and W. Kamakura (1998), *Market Segmentation : Conceptual and Methodological Foundations*, International Series in Quantitative Marketing, Kluwer Academic Publishers.

Wedel, M. and W. Kamakura (2002), Introduction to the Special Issue on Market Segmentation , *International Journal of Research in Marketing*, **21**, 181–183.

Wikipedia (2006), `http://en.wikipedia.org`.

Wilson, K. (1979), Problems in Physics with Many Scales of Length, *Scientific American*, **241**, 158–179.

Wolfe, P. and L. Cutler (1963), Experiments in Linear Programming, in R. Graves and P. Wolfe (eds.), *Recent Advances in Mathematical Programming*, McGraw Hill, pp. 177–200.

Yavas, U., B. Verhage, and R. Green (1992), Global Consumer Segmentation versus Local Market Orientation, *Management International Reviews*, **32**, 265–273.

Yeo, A. (1997), *Large Exponential Neighbourhoods for the TSP*, Ph.D. thesis, University of Odense.

Yorke, D. (2001), The Definition of Market Segments for Leisure Centre Services: Theory and Practice, *European Journal of Marketing*, **18**, 100–113.

Zhang, W. (1993), Truncated Branch-and-Bound: A Case Study on the Asymmetric TSP, in *Proc. AAAI -93 Spring Symposium on AI and NP-Hard Problems*, Stanford, pp. 160–166.

Zhang, W. (2000), Depth-First Branch-and-Bound versus Local Search: A Case Study, in *Proc. 17th National Conference on Artificial Intelligence (AAAI-2000)*, Austin, Texas, pp. 930–935.

Zhang, W. (2003), Phase Transitions of the Asymmetric Traveling Salesman Problem, in *Proceedings 18th International Joint Conference on AI (IJCAI-03)*, Acapulco, Mexico, pp. 1202–1207.

Zhang, W. (2004), Phase Transitions and Backbones of the Asymmetric Traveling Salesman Problem, *Journal of Artificial Intelligence Research*, **20**, 471–497.

Zhang, W. and R. Korf (1996), A Study of Complexity Transitions on the Asymmetric Traveling Salesman Problem, *Artificial Intelligence*, **81**, 223–239.

Zhang, W. and J. Pemberton (1994), Epsilon-transformation: Exploiting Phase Transitions to Solve Combinatorial Optimization Problems – Initial Results, in *Proceedings of AAAI-94*, Seattle, WA.

Index

Samenvatting (Summary in Dutch)

Dit proefschrift gaat over moeilijk oplosbare wiskundige problemen die veelvuldig in de praktijk opduiken. Ze behoren tot de klasse der *combinatorische optimalisatieproblemen* (COPs). Een kenmerk van dergelijke problemen is dat er een eindig aantal mogelijkheden is waaruit een optimale keuze bepaald dient te worden. Met 'optimaal' bedoelen we gewoonlijk dat de kosten zo laag mogelijk dienen te zijn, of de winst zo hoog mogelijk. Een voorbeeld van een COP is het kortste pad-probleem, dat in routeplanners opgelost wordt: gegeven een aantal verbindingen tussen locaties wordt getracht een kortste pad tussen de vertrekplaats en de bestemming te bepalen. De keuzemogelijkheden zijn hier de mogelijke paden tussen het vertrek- en aankomstpunt en dat zijn er heel erg veel!

De verzameling mogelijkheden heet de *zoekruimte* van het probleem. Ook al bestaat de zoekuimte uit een eindig aantal mogelijkheden, dat betekent nog niet dat het probleem opgelost kan worden door successievelijk alle mogelijkheden te evalueren. Vaak zijn er voor COPs buitensporig veel keuzemogelijkheden. Voor veel praktische problemen bestaan evenwel snelle oplossingsmethodes, zoals voor het kortste pad-probleem. Dergelijke problemen worden normaal gesproken binnen enkele secondes opgelost, ook al zijn er honderden miljoenen mogelijkheden. Dit komt doordat slimme oplossingsmethodes slechts een fractie van alle mogelijkheden in beschouwing nemen.

Voor een grote klasse problemen bestaan zulke slimme methodes echter niet en zullen ze, naar velen vermoeden, ook nooit gevonden worden. In het slechtste geval dienen bijna alle mogelijkheden in de zoekruimte bekeken te worden, voordat een optimale oplossing vastgesteld kan worden. Problemen in deze klasse worden $\mathcal{NP}$-lastig genoemd ($\mathcal{NP}$-*hard* in het Engels).

Het vinden van een optimale oplossing voor een $\mathcal{NP}$-lastig probleem kan dus zeer veel tijd vergen. Indien die tijd niet beschikbaar is, neemt men in de praktijk vaak de toevlucht tot *heuristieken*: snelle methodes die niet noodzakelijk tot optimale oplossingen leiden. Hoewel met zogenaamde *meta-heuristieken* oplossingen bepaald kunnen worden die hooguit slechts een paar procent slechter zijn dan de optimale, kan een extra verbetering van een paar procent tot besparingen leiden die zeer de moeite waard zijn. Dit feit motiveert in belangrijke mate de zoektocht naar verbeterde oplossingstechnieken.

Een belangrijke klasse van oplossingsmethodes voor $\mathcal{NP}$-lastige problemen is *Branch and Bound (BnB)*. Veel $\mathcal{NP}$-lastige problemen worden gemakkelijk oplosbaar wanneer er extra restricties opgelegd worden. De ontstane gemakkelijk oplosbare problemen heten *relaxaties*. BnB methodes beginnen met het oplossen van een relaxatie. Als de gevonden oplossing niet *toelaatbaar* is, dat wil zeggen, als het niet tot de zoekruimte van het oorspronkelijke probleem behoort, dan wordt de zoekruimte opgedeeld in *subproblemen*. Dit proces gaat door totdat alle subproblemen ofwel opgelost of uitgesloten zijn. Een subprobleem is *opgelost* als de oplossing van de relaxatie toelaatbaar is voor het oorspronkelijke probleem; een subprobleem heet *uitgesloten* als het geen verdere bijdrage levert aan het oplossingsproces.

Hoodstuk 2 gaat over een aanpassing van de BnB techniek voor het zogenaamde *asymmetrische handelsreizigersprobleem*. Dit probleem duiden we aan met de aan het Engels ontleende afkorting *ATSP*. Gegeven n ($n \geq 3$) locaties en de afstanden tussen elk paar locaties, dient er een kortste rondrit, ofwel een kortste complete *tour*, gevonden te worden die elke locatie precies één keer aandoet. De afstand tussen locatie i en locatie j is hierbij niet noodzakelijkerwijs gelijk aan de afstand tussen j en i. Het ATSP is $\mathcal{NP}$-lastig en BnB technieken zijn veel gebruikte methoden om een optimale oplossing van het ATSP te vinden. De meest voor de hand liggende relaxatie is het toewijzingsprobleem, afgekort met AP. In het AP dient elke locatie in een tour te liggen, maar hiervoor mogen meerdere gescheiden tours, de zogenaamde *subtours*, gebruikt worden. De gangbare techniek om een ATSP-oplossing te maken uit een AP oplossing met meer dan één subtour is *patching*. De subtours worden hierbij samengevoegd tot één complete tour door alle locaties. De kosten die gepaard gaan met patching dienen als bovengrens voor de optimale ATSP-tour wanneer deze lager zijn dan de waarde van de beste oplossing tot dan toe in het BnB proces. Als we kunnen vaststel-

len dat alle toelaatbare oplossingen van een subprobleem hogere kosten hebben dan de huidige beste oplossing, dat wil zeggen, als de benedengrens van het subprobleem hoger is dan de huidige bovengrens, dan kunnen we dit subprobleem uitsluiten. Hoe beter de bovengrens, des te kleiner is het aantal subproblemen dat de BnB methode in beschouwing hoeft te nemen.

In Hoofdstuk 2 wordt *iterative patching* geïntroduceerd; een methode waarin patching toegepast wordt in elk subprobleem in de BnB zoekprocedure. In bestaande BnB methodes voor het ATSP, zoals in Carpaneto *et al.* (1995), wordt patching alleen toegepast bij initiële subproblemen. Deze methodes maken gebruik van een *Best First Search (BFS)* strategie, een zoekstrategie waarin het meest veelbelovende subprobleem eerst opgelost wordt. BFS heeft het nadeel dat er veel geheugen nodig is om moeilijke probleeminstanties met veel subproblemen op te lossen. Daarom gebruiken wij *Depth First Search (DFS)*, een zoekstrategie waarbij het meest recentelijk gegenereerde subprobleem als eerste opgelost wordt. Het blijkt dat, in geval van *DFS*, *iterative patching* zinvol is. Het uitsluiten van subproblemen leidt tot een zodanige tijdwinst dat de tijd die nodig is om de patching uit te voeren ruimschoots vergoed wordt. De *KSP-procedure* van Karp en Steele (1991) en de *Contract-or-Patch* heuristiek van Gutin *et al.* (2001) blijken de beste patching-procedures om in iterative patching te gebruiken.

In Hoofdstuk 3 worden een aantal andere onderdelen van BnB methodes bekeken, wederom voor het ATSP en de AP-relaxatie. Indien een BnB methode als zoekstrategie *DFS* gebruikt, dan betekent dit dat de methode eerst een subprobleem geheel uitpluist, voordat een ander subprobleem in beschouwing genomen wordt. In het geval van het ATSP en de AP-relaxatie wordt een verbinding, zeg e, verboden. Daarna worden eerst subproblemen opgelost waar de verbinding e niet in voorkomt. Als e in elke optimale ATSP-oplossing voorkomt, dan heeft het BnB algoritme tevergeefs veel tijd besteed aan het zoeken in dit deel van de zoekruimte. Het is dus belangrijk om goed te voorspellen welke verbindingen gehandhaafd dienen te blijven, de *overlevers*, en welke verwijderd moeten worden, de *uitstervers*.

BnB methodes met de DFS-zoekstrategie bepalen de verbinding die verwijderd moet worden als volgt. Zoek de kleinste subtour (met het kleinste aantal verbindingen) en verwijder de verbinding met de hoogste kosten. Wij stellen voor om, in plaats van kosten, *boventoleranties* te gebruiken. De boventoleranties van

de verbinding e met betrekking tot de AP-oplossing is de maximale toename in de kosten van de verbinding waarbij deze zich nog steeds in deze optimale AP-oplossing bevindt. Onze experimenten tonen aan dat met boventoleranties accurater bepaald kan worden wat de overlevers en de uitstervers zijn, veel beter dan met kosten. We tonen ook aan dat de *benedengrens* van een subprobleem kunnen verhogen door boventoleranties te gebruiken. De benedengrens van een subprobleem is ten hoogste de waarde van de goedkoopste toegelaten opklossing van een subprobleem. Als deze waarde verhoogd wordt, hoeven minder subproblemen in beschouwing genomen te worden. Beide verbeteringen leiden ertoe dat het aantal te bestuderen subproblemen aanzienlijk teruggebracht kan worden. Wanneer beide verbeteringen samen geïmplementeerd worden, zijn de reducties groter dan de totale reductie wanneer de verbeteringen apart geïmplementeerd worden: het *synergie-effect*.

Toleranties hebben het nadeel dat het tijd vergt om ze te berekenen: om één boventolerantiewaarde te bepalen, dient een geheel AP opgelost te worden (al kan dit wel versneld uitgevoerd worden). De experimenten tonen desondanks aan dat voor veel instanties, inclusief instanties uit de praktijk, onze op toleranties gebaseerde methodes sneller werken dan de op kosten gebaseerde methodes. Ook blijkt dat deze DFS-methodes zich kunnen meten met de BFS-methode van Carpaneto *et al.* (1995). Met name voor moeilijke instanties kan de laatstgenoemde methode vaak geen oplossing vinden binnen de gestelde tijdslimiet van één uur, terwijl onze DFS-methodes daartoe wel in staat zijn.

In Hoofdstuk 4 wordt aanvullend onderzoek gedaan naar op toleranties gebaseerde BnB algoritmes. Ten eerste beschouwen we met behulp van toevalsgetallen gegenereerde instanties voor het ATSP. Op toleranties gebaseerde BnB methodes zijn met name snel voor problemen met veel verschillende afstanden. Deze problemen zijn volgens Zhang (1993) de moeilijkste toevalsinstanties. Ten tweede trachten we onze BnB-methodes te verbeteren door toleranties alleen te gebruiken 'boven' in de zoekboom; we verwachten dat ze hier het meest effectief zijn. Experimenten tonen echter aan dat dit niet altijd waar is. Ten derde beschouwen we de mogelijkheid om niet één, maar meerdere tolerantiewaarden bij de benedengrens op te tellen. Voor het AP blijkt dat niet mogelijk te zijn, maar voor het zogenaamde *Minimum Spanning Tree Probleem* wel; dit tonen we aan in Sectie 4.4

Hoofdstuk 5 tenslotte past de kennis van de hierboven besproken metho-

dieken toe op een concreet probleem uit de praktijk: het *segmentatieprobleem*. Hierin worden groepen consumenten, de zogenaamde *segmenten*, onderscheiden die min of meer op dezelfde wijze reageren op marketinginspanningen van een bedrijf. Bijvoorbeeld, consumenten in één segment kopen meer van het product tijdens een promotie-actie, terwijl andere consumenten hun koopgedrag nauwelijks veranderen. Wij concentreren ons hier op segmentatieproblemen waarbij de onderneming verschillende segmenten identificeert en vervolgens aan ieder segment een marketingmix toewijst.

Met name in internationale markten en bij bederfelijke goederen kunnen de logistieke kosten zo hoog worden dat bedrijven niet in staat zijn om segmentatiestrategieën, die het resultaat zijn van gebruikelijke methodes, uit te voeren. Daarom vallen ze terug op een strategie waarin landen of regio's als aparte segmenten bediend worden; zie Steenkamp en Ter Hofstede (2002). Een dergelijke strategie heeft als nadeel dat verschillen binnen landen en overeenkomsten tussen consumenten in naburige landen niet benut kunnen worden. In Hoofdstuk 5 geven we een methode die een afweging maakt tussen logistieke kosten enerzijds en het maken van zo homogeen mogelijke segmenten, de *fit*, anderzijds. Toepassing op een specifieke segmentatiestudie toont aan dat deze benadering in staat is internationale segmenten met relatief lage logistieke kosten te construeren. De segmentaties worden bepaald met behulp van *simulated annealing*, een effectieve en moderne metaheuristiek, die voor deze problemen goede oplossingen levert.

Tenslotte bestuderen we de mogelijkheid de BnB-methode uit Diehr (1985) toe te passen op de segmentatieproblemen uit onze cases. Het blijkt dat deze BnB methode niet snel genoeg is om segmentatieproblemen met 123 elementen op lossen. Het blijkt zelfs meestal niet mogelijk de oplossing verkregen met simulated annealing te verbeteren.